National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

UNIVERSITY OF ALBERTA

A Hierarchical Model for Virtual Environments

BY

Yunqi Sun ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

DEPARTMENT OF COMPUTING SCIENCE

Edmonton, Alberta
Fall 1995

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-612-06299-6

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Yunqi Sun

TITLE OF THESIS: **A Hierarchical Model for Virtual Environments**

DEGREE: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1995

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

(Signed) Yunqi Sun

Yunqi Sun
1–202, Building 43
Nan Shan Residence
Dalian University of Technology
Dalian, Liaoning Province
The People's Republic of China
116024

Date: Sept. 14th, 1995

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A Hierarchical Model for Virtual Environments** submitted by Yunqi Sun in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Dr. M. Green (Supervisor)

Dr. S. Greenberg (External)

H. James H.

Dr. H. J. Hoover (Examiner)

Peter v. Beek

Dr. P. van Beek (Examiner)

Dr. B. Cockburn (Examiner)

Dr. J. Culberson (Chair)

Date: Sept. 14th, 1995

To my parents

Abstract

It is difficult to build large, complicated virtual environments with many application objects. There are a number of reasons for this, and this thesis concentrates on three of them. First, the response time increases as the number of objects increases. Second, for multi–user virtual environments, the users should be able to join or quit from the shared environment at any time. Third, in order to ensure a consistent view for all users, their databases must be synchronized, which may produce substantial network traffic.

This thesis proposes a new architecture for shared virtual environments, describes a prototype implementation and the evaluation of this architecture. A hierarchical Virtual Object Model is proposed to help programmers analyze, design and implement applications from a high level to a more detailed level. A hierarchical Virtual Place Model is introduced to organize virtual environments and to put application objects into different sub–environments. Only the objects that are in the sub–environments that are visible to the user are evaluated and presented. Consequently, the response time is greatly improved since it is only proportional to the complexity of the user's current sub–environment, rather than the complexity of the whole application environment. For multi–user applications, a communication mechanism is provided to automatically update databases for late–comers. Local changes are only sent to the users who can see the changes, so the network traffic generated by the local user is proportional to the number of cooperating users, as opposed to all the users in the entire virtual environment.

Acknowledgments

I would like to thank my supervisor Professor Mark Green for his guidance, encouragement, patience and support throughout the course of this thesis research. I also like to thank Professor Jim Hoover and Professor Peter van Beek for carefully reading this thesis and providing many insightful comments. The time and effort that Professor Bruce Cockburn and Professor Saul Greenberg spent reviewing and evaluating this thesis is truly appreciated. I would like to acknowledge the Computing Science Department for providing some of the financial support in the last five years. Special thanks go to Chris Shaw and Haiying Wang for their helpful suggestions during the implementation of my thesis project. Thanks also go to Robert Lake for providing a convenient working environment in our graphics lab. I wish to thank Hunqiu Sun, Jiandong Liang and Qunjie Wang for their interesting discussions and helpful comments. I am very grateful that my husband, Weiye Zhang, has always been very understanding and patient since the beginning of my Ph.D study. Lastly, I would like to thank my parents. Without their love and support, this dissertation would not have been completed.

Contents

1 Introduction **1**

1.1 Terminology and Motivation 1

1.2 The Problem 2

1.2.1 The Geometric Model Component 5

1.2.2 The Computation Component 6

1.2.3 The Interaction Component 6

1.2.4 The Presentation Component 9

1.2.5 Communication Issues 9

1.3 Research Goals 10

1.3.1 VR System Model 11

1.3.2 VR Environment Structure 11

1.3.3 Navigation Facilities 12

1.3.4 Access to Other Sub-Environments 13

1.3.5 Better Response Time 13

1.3.6 Multi-User VR Systems 14

1.3.7 Lower Network Traffic 15

1.4 Thesis Outline 15

2 Background **17**

2.1 VR Applications 17

2.1.1 Visualization 17

2.1.2 Training 19

2.1.3 Computer–Aided Design 19
2.1.4 Telecommunications 20
2.1.5 Entertainment 20
2.1.6 Discussion 20
2.2 VR Devices 21
2.2.1 Criteria for VR Devices 21
2.2.2 Visual Display Devices 22
2.2.3 Auditory Display Devices 26
2.2.4 Haptic Display Devices 28
2.2.5 Interaction Devices 31
2.2.6 Discussion 32
2.3 VR Software Development 33
2.3.1 System Configuration 33
2.3.2 Single–User Systems 34
2.3.3 Multi–User Systems 35
2.3.4 Concurrency Control 38
2.3.5 VR Environment Organization 41
2.3.6 Discussion 41

3 Virtual Environment Model **43**
3.1 Virtual Object Model 44
3.2 The Place Model 47

4 Multi–User VR System Architecture **52**
4.1 The VR Session Manager 55
4.2 The Communication Component 56
4.3 The Geometric Model Component 58
4.4 The Interaction Component 59
4.5 The Computation Component 59
4.6 The Presentation Component 60

5 Multi–User Place System 62
5.1 System Overview . . . 62
5.2 Class Hierarchy . . . 65
5.3 Object Class . . . 66
  5.3.1 Physical Properties . . . 66
  5.3.2 Structure . . . 68
  5.3.3 Behavior Evaluation and Presentation . . . 70
5.4 Link Object Class . . . 71
  5.4.1 Transformation Matrix . . . 71
  5.4.2 Animation Behavior Evaluation . . . 72
5.5 Place Class . . . 72
  5.5.1 Physical Properties and Structure . . . 72
  5.5.2 Behavior Evaluation and Presentation . . . 75
5.6 Door Class . . . 75
  5.6.1 Navigation Related Functions . . . 76
  5.6.2 Visible Places' Behavior Evaluation and Presentation . . . 77
5.7 Crystal Ball Class . . . 78
5.8 User Class . . . 79
5.9 Shared User Class . . . 81
5.10 Communication Agent Class . . . 82
  5.10.1 Shared User Management . . . 83
  5.10.2 Shared Data Management . . . 84
  5.10.3 New Protocols . . . 87
  5.10.4 Transaction Management . . . 88
  5.10.5 Request Processing . . . 88
5.11 The Session Manager . . . 89
  5.11.1 Shared User Management . . . 89
  5.11.2 Transaction Management . . . 90
5.12 Extending Existing Computer Technologies . . . 91

5.12.1 The Filesystem Model as an Analogy to the Place Model . . . 91
5.12.2 Extending the Traditional User Interface Model 94
5.13 Summary 96

6 Evaluation **97**
6.1 Performance Analysis 97
6.1.1 Response Time Improvement 98
6.1.2 Network Traffic Reduction 108
6.2 Application Example 115
6.2.1 Multi–User Place System Application 116
6.2.2 Response Time Improvement 118
6.2.3 Network Traffic Reduction 124
6.3 Summary 128

7 Conclusion **131**
7.1 Contributions 131
7.2 Future Work 133
7.2.1 Security Problem 133
7.2.2 Concurrency Control 133
7.2.3 System Bottleneck 134
7.2.4 Traveller Problem 134

A Example Application Programs **136**
A.1 Program of Example Environment Construction 136
A.2 Program of Class VClock 139
A.2.1 Aclock.h 139
A.2.2 Aclock.C 140
A.3 Program of Class AObject 142
A.3.1 Aobject.h 142
A.3.2 Aobject.C 143

List of Figures

3.1 The Virtual Object Model 46
3.2 The Place Model 48

4.1 Multi-user VR system architecture 54

5.1 Multi-User Place System 63
5.2 Multi-User Place System Class Hierarchy 65
5.3 Analogy between Filesystem model and Place model 91
5.4 Correspondence between all types of files and VOs 92
5.5 Traditional software system model 94
5.6 VR software system model 95

6.1 Hierarchical structure of the application environment 117
6.2 Hierarchical structure of the clock object 118
6.3 Compare userA's results in both applications 121
6.4 Compare userB's results in both applications 122
6.5 Communications among SM, userA and userB in NoPlaceApplication 125
6.6 Communications among SM, userA and userB in PlaceApplication . . 127

List of Tables

6.1 Test results for userA in the NoPlaceApplication 120
6.2 Test results for userB in the NoPlaceApplication 120
6.3 Test results for userA in the PlaceApplication 120
6.4 Test results for userB in the PlaceApplication 121
6.5 Compare the test results of userA in NoPlaceApplication and in PlaceApplication 123
6.6 Compare the test results of userB in NoPlaceApplication and in PlaceApplication 123
6.7 Transactions occured at userA's site in NoPlaceApplication 124
6.8 Transactions occured at userB's site in NoPlaceApplication 125
6.9 Transactions occured at userA's site in PlaceApplication 127
6.10 Transactions occured at userB's site in PlaceApplication 127

Chapter 1

Introduction

1.1 Terminology and Motivation

Virtual reality is a new field of computer science, which so far has no formal definition. But the following description gives the necessary characteristics of a virtual reality system:

"Virtual reality is a three–dimensional computer–generated environment with which the user can dynamically interact."

Virtual reality is also called virtual environment, virtual world, artificial reality, artificial environment, or cyberspace.

In this thesis, we use the following terms:

Virtual Reality (VR) refers to the new technology used to produce three–dimensional interactive environments.

Virtual Environment (VE) refers to an environment produced by a virtual reality system. It is a new type of graphical user interface, compared to traditional 2D user interfaces.

Virtual World (VW) is a metaphor for Virtual Environment. Usually, a Virtual Environment is a simulated real world or representation of information in the form of the physical world.

VR technology has many potential applications, including visualization, training, computer–aided design, telecommunications, and entertainment. The future of virtual reality is promising. However, virtual reality technology is still relatively immature. Producing a large and complicated VR environment is difficult. The objective of this thesis is to explore VR techniques that provide high–level support for designing and implementing multi–user VR applications which have better response time and lower network traffic. The final goal is to build a prototype software system to support the development of such application systems.

1.2 The Problem

There are three essential requirements for a VR system to be successful, or at least acceptable: response time, realism, and usability. When an application environment with a large number of application objects is constructed, quick response time is not easy to achieve. As the number of objects increases, object behavior evaluation time and rendering time increase proportionally. Consequently, the response time will be unbearable for the users.

One possible solution to the problem is to divide the application environment into several sub–environments, and put the application objects into different sub–environments. The behavior evaluation time and rendering time of those objects that are not visible to the user can be saved , thus improve the response time. However, the usability could be impaired by the sub–environment structure if the user needs to operate on an object in another sub–environment, or observe what is going on in several other sub–environments while working in the current sub–environment.

Another issue concerning usability is whether a VR system allows multiple users to share the application environment, and to enter or leave the environment as they wish. In order to provide the same view of the shared environment to all users, the system must somehow synchronize the users' databases. The synchronization could produce substantial network traffic among shared users. The network delay, which is the time between one user's action and when the result reaches another user's site, would be longer.

Usually, a VR system is composed of input/output (I/O) devices, supporting software, and the application. All these parts of a VR system have to meet the aforementioned requirements to generate a responsive, realistic, easy–to–use Virtual Environment.

VR technology involves a few new hardware devices that introduce new channels of information exchange between computers and users. The new devices include the DataGlove, EyePhone, HeadPhone, and devices for force–feedback. Researchers at NASA, the University of North Carolina at Chapel Hill [13, 27, 44, 30, 40] and other institutes have been working to upgrade the performance of these devices.

To build a VR application still requires a large amount of time and effort. High–level VR support software is needed to assist the programmer in analyzing, designing, and implementing VR application systems. To improve the performance of an application is not easy either. Nevertheless, there are many application–dependent methods to improve performance. This is largely the responsibility of the application programmer. Performance evaluation tools are certainly helpful. At the University of Alberta, we use a Timing software package to measure the response time of VR software systems [36].

The VR support software that lies between the hardware devices and the application level is the major concern of this thesis. VR support software should provide mechanisms to improve response time for any VR application. For VR devices, realism means minimum distortion of the presented information. For VR support software, animation feedback should be supported to give the user a sense of realism. VR support software should also strongly support VR system usability. A virtual environment should be structured so that it is easy for the users to investigate it. It should also provide powerful tools for the user to interact with objects within an environment as well as interact with other users. In this context, response time, environment structure, interaction, and multi–user problems will be discussed.

Zeltzer et al. [46] suggested a constraint network control structure for building interactive simulation systems, especially for task–level animation systems. Robertson et al. [34] designed the Cognitive Coprocessor Architecture to support "multiple, asynchronous, interactive agents" and smooth animation. But these two models did not address the response time of VR systems. Furthermore, they were designed for single user systems, not multi–user systems.

IBM researchers use a centralized approach to implement a multi-user VR simulation system [8]. In their system, each user has his or her own interface style but shares the underlying simulation with all other users. The centralized approach is simple. However, a replicated approach has the advantage of better response time and lower network traffic without losing user–specific interaction style, especially for users who are several miles away from each other.

The SIMNET system [5] developed by the LORAL Advanced Distributed Simulation Inc and the DIVE system [7] built at Swedish Institute of Computer Science both use a replicated architecture, but the changes are broadcast to all users, although SIMNET uses the "dead reckoning" technique to extrapolate moving objects' states so as to reduce network traffic. However, SIMNET does not structure the environment; whereas in DIVE the whole virtual environment is a collection of different "worlds" which contain different sets of objects and users can switch worlds dynamically. The response time problem is not considered by the two systems.

The WAVES [21] and BrickNet [37] systems are similar in that both of them allow different users to have different virtual world contents while still sharing some objects. The updated data for a shared object are sent only to the users who have the object. BrickNet also provides a "dynamic portal" for a user to go to another world, select and copy objects from that world and bring them back to the user's local world. These two systems did not attack the response time problem directly, but they do improve the response time because only the user's local world is evaluated and displayed to the user. However, if the user's local world contains too many application objects, the response time problem still exists.

At the University of Alberta, the Decoupled Simulation Model [36] has been proposed. A VR application is broken into four components: the Interaction component, the Geometric Model component, the Computation component, and the Presentation component. The VR system consists of one or more processes which can be distributed to different machines therefore speeding up the whole system. The Minimal Reality (MR) Toolkit was built to support this model. The MR Toolkit has device-level software, which also provides support for realism (such as filter procedures), and several software packages that support interaction techniques are also provided.

The MR Toolkit addresses low-level, device-related issues. It provides little support for application environment design and multi-user application design. The software packages in MR facilitate the Interaction component and the Presentation component of an application. The Geometric Model component is totally up to the VR application programmer to design and construct. In addition, the Decoupled Simulation Model was designed for single user applications, not for multi-user applications. When we take application level software and multiple users into account, a lot of problems still need to be solved.

1.2.1 The Geometric Model Component

The Geometric Model component maintains a high-level representation of the data in a virtual environment. When a large, complicated application environment is constructed, it is difficult to manage the data representation. A model for describing and organizing virtual environments must be developed to help in the design and implementation of applications.

Bricken et al. at the University of Washington have suggested a "Virtual Environment Operating System" (VEOS) to manage the data structure. The idea is that everything in an virtual environment is an Entity. They did not address the response time problem.

Card et al. [6] introduced 3D Rooms to organize information workspaces so that the information frequently needed, or in immediate use, is in the current Room. Only the current Room is displayed. The user can walk through doors into other Rooms to switch from one cluster of information to another. The overall cost of information retrieval is lowered. This idea can also be used to organize virtual environments. However, 3D Rooms is a 3D graphics system, not a virtual reality system. In a virtual reality system, the navigation problem is much more complicated, as will be discussed in the context of interaction. As well, there is no way the user can access information in another Room from the current Room. Consequently, this 3D Room structure may impair the usability of the system.

1.2.2 The Computation Component

The Computation component updates the data in the Geometric Model component. The data sharing package in the MR Toolkit facilitates data communication. Since we use two graphics workstations to display pictures to the user's two eyes, the data in the Geometric Model components on both machines has to be synchronized. The same problem appears when more than one user shares a virtual environment at different sites. However, not all the users will work in one sub–environment all the time. Sometimes, the result of one user's action will not be visible to another user for quite a long time. In this case, sending the results through the network only increases network traffic.

1.2.3 The Interaction Component

The Interaction component manages input from the user. The MR Toolkit has several software packages to support the development of the Interaction component. However, the construction of some portions of this component is not fully supported, especially when we consider structured VR environments and multi–user environments.

The current MR Toolkit does not have a good filter for the DataGlove. The jittering problem makes it difficult to do actions such as selecting. Varying the control–to–display (C/D) ratio is helpful. Unfortunately, the DataGlove is an absolute device rather than a relative device like the mouse. If the accumulated distance between the real position of the hand, and the drawn position of the DataGlove becomes too large, the user will no longer feel that it is his or her hand that is wearing the glove.

The DataGlove gesture editing program in the MR Toolkit allows the programmer or the end user to interactively define the gestures to be used in applications. But the number of gestures that we can think of is very limited. Consequently, applications are not able to provide the user with flexible and powerful interfaces. The Panel package of the MR Toolkit can be used to produce 3D menus, which could help to enlarge the user's command vocabularies. But it is difficult to either select a menu item or confirm the selection by the Dataglove because of the jittering problem.

The MR Toolkit has a Workspace Mapping package. The new VR devices, including the EyePhone, DataGlove, and 3D sound, all have their own coordinate systems, which are dependent on the device locations. The Workspace Mapping package converts the device coordinates to room coordinates and then to virtual environment coordinates. Thus, the coordinates of devices which the programmer receives are the coordinates in the virtual environment. However, if the virtual environment is organized into a structured environment, the coordinates of devices should be transformed into the sub–environment that the user is occupying.

Usually, the Interaction component is designed to process all inputs to the environment in the order of their arrival. However, if the result of the current event will not be immediately visible to the user, and there are other requests with higher priority, a better update rate can be obtained by using a scheduling algorithm. For instance, when the user is dragging an object in the current Room, a clock shows the updated time in another Room. The Computation component should process the dragging request first, and leave the time update until later, or even skip it. The criteria for scheduling requests are correctness and efficiency. Only when the suspended request is processed before its results are used, can correctness be guaranteed. If the scheduling algorithm is too complex, the response time will degrade rather than improve.

In a structured VR environment, the user usually works in the current sub-environment. But there are cases when the user needs access to an element which is not in the current sub-environment. For example, a user may need to know the time from a clock which is in another sub-environment, or may need to set the time for that clock. In other cases, the user may have to observe what is happening in several other sub-environments, while working on something else in the current sub-environment. The environment structure, while meant to help improve response time, should not become an obstacle to the application system's usability.

Navigation is always an issue in virtual reality. In practice, the virtual environment is most likely larger than the physical space the user can walk around. The user must be able to travel to any point within the virtual environment without physically walking that distance. Flying is a popular navigation tool in most VR systems. The user can "fly" freely within the whole virtual environment as if he or she is driving a helicopter. Flying is a powerful navigation aid if the virtual environment is reasonably small, or the user is quite familiar with the setting of the virtual environment. However, if the user is not familiar with the environment, it is very easy to get lost. Landmarks can be of great help during the navigation process. Again, the user must know the surrounding environment quite well to figure out how to get to where he or she wants to go from his or her current location. Otherwise, landmarks will not make any sense to the user.

The navigation problem becomes even more complicated if we have many sub-environments in a single large VR environment. Navigation facilities must be provided to allow the user to move from the current sub-environment to other sub-environments. Animation feedback is a very important feature of any navigational aid. If the user is suddenly transported from the current sub-environment to another one, the user will probably be confused and disoriented. As the user crosses the border between two sub-environments, the user's coordinates should be in the coordinate system of the new sub-environment instead of the previous sub-environment.

Usually, every sub-environment has a boundary to remind the user of the limit of his or her current working place. For instance, a virtual room has its surrounding walls as its boundary. As long as the user always remains within the boundary of the current sub–environment and can walk only through doors to go to other sub–environments, the chance that the person will get lost is small. Unfortunately, these boundaries of the sub–environments are virtual, not real boundaries. They can not stop the user from walking through them as real walls do. Unless the user is completely familiar with the locations and setting of every sub–environment, he or she will most likely become lost if he or she unintentionally walks through the boundary of the current sub–environment. Help information about the user's location should also always be available, too.

1.2.4 The Presentation Component

The Presentation Component presents the synchronized visual, sonic and haptic (force feedback or tactile feedback) display to the user. Usually, every element in the VR environment will be presented to the user, since it is hard to determine the elements that are not visible. But rendering something invisible is a waste of time and degrades response time. An alternative is to display only the current sub–environment which the user is in, that is, to assume that all doors leading to other sub–environments are always closed. However, this is counter–intuitive in that the user can not see elements in another sub–environment even when he or she is walking through an open door to it. Can we automatically display the current sub–environment with open doors to other sub–environments?

1.2.5 Communication Issues

When we construct a multi–user VR application, the data sharing package of the MR Toolkit can be used to deliver messages among users. But coordinating communications among users, and managing the entrance or departure of users from the environment is beyond the functionality of the current data–sharing package. Whenever a late–comer joins in a shared virtual environment, the local database must be updated so it is consistent with the other users' evolving databases, even after the user who made the changes has left the shared environment. Communication protocols must be designed for maintaining the changes to the database and updating the databases for shared users whenever necessary. The naming mechanism of the data sharing package needs to be improved, too. For now, every shared data gets an internal name in the order of its declaration. When two users dynamically create two different objects to be shared at the same time, the two objects will get the same internal name at their local machines. If the internal name has to be obtained from a centralized location such as a manager, different users working on different objects will have to compete for the internal names. This will introduce a bottleneck into the system and degrade system performance.

1.3 Research Goals

The objective of this thesis is to explore models and related techniques to provide high–level support for VR programmers to design and implement high–performance, multi–user VR application systems. The following goals will help achieve the objective:

- To propose a VR system model which could help the programmer analyze, design and implement application systems from a high level to a more detailed level.
- To propose a VR environment structure to organize large, complicated VR environments into sub–environments. Related elements can be put into the same sub–environment.
- To provide navigation facilities in the environment structure.
- To provide mechanisms to enable access to non–current sub–environments.
- To improve response times of VR systems.

- To provide support for multi–user VR systems and database updates when a late–comer joins a group of geographically distributed users in a shared virtual environment.
- To reduce network traffic among shared users.

1.3.1 VR System Model

Typically, a VR application system receives input from users, processes the input, updates and synchronizes databases, and presents output to the users. The output will show the users how the virtual environment reacts to the input, and how it changes itself even without user interference. A virtual environment is typically composed of many virtual objects. How the virtual environment reacts to the input and how it changes itself are actually a result of how each object, or parts of the objects, reacts to the input and changes. The output presented to the users is a collection of objects with their own physical attributes. My goal is to decompose VR application design and implementation into various levels of object design and implementation. The VR support software assembles all the objects into the complete application system. As a result, the programmer productivity can be greatly increased.

1.3.2 VR Environment Structure

People tend to classify their working objects into different categories. Usually, they work on one set of objects, and after a certain period of time, switch to another set of objects. A large object set can be split into several smaller object sets with more closely related objects in the same set. We support different sets of objects placed in different levels of sub–environments in the virtual world. Every sub–environment may have a boundary to separate itself from the rest of the environment. This environment structure provides the foundation for response time improvement and network traffic reduction.

1.3.3 Navigation Facilities

One of the primary goals of virtual reality is to allow users to make use of their daily experience in real life to live and act in a virtual world. Users can move around within the virtual world by physically walking or turning around. However, a user can not walk beyond the VR devices' sensing limit or the boundary of the room in which the user is physically located. Navigational facilities must be provided to allow users to move from their current locations to any other locations within the virtual environment with minimum effort.

Flying, as a navigational aid, allows users to move around in a large virtual environment such as an open field. Users can use their driving experience to navigate themselves in the virtual world. But people do not have a ghost's experience of going through walls in their real lives. Even though this experience could be exciting, it might be quite confusing to most people. Unless they are really familiar with the structure and setting of the environment, they will not be able to tell where they are, how to get back where they were, and how to get where they would like to go. In real life, people do have a lot of experience going through doors to go to a particular room in a big building. Since virtual environments are organized into different levels of sub–environments, doors are provided to facilitate navigation between any two sub–environments. Animation feedback is also provided. The door will gradually open. The user is slowly transported to the other side of the door, just as in real life. Once a user enters a new sub–environment, the user's coordinates will be changed to the coordinate system of the current sub–environment, to allow the user to interact with all the objects within the new sub–environment.

Other facilities can also be provided to offer help information or to prevent users from getting lost. Every sub–environment can be built with a landmark in it. The name of the sub–environment can be shown on a name tag somewhere within the sub–environment. The boundary of a sub–environment can also be set to move away whenever the user tries to walk through it. Help information about the user's current location is always available. The path from the user's current sub–environment to the top level environment — the virtual world — should remind the user of his or her current position within the entire virtual environment.

1.3.4 Access to Other Sub–Environments

Most of the time the user works only on the objects in the current sub–environment. But sometimes the user may need to reference an object that is not in the current sub–environment. Sometimes several other sub–environments have to be observed at the same time. The proposed system provides copies of objects and sub–environments to meet such requirements. The copy of a remote object can be used to avoid navigating back and forth between different sub–environments. The user can view and manipulate the copy just as he or she views and manipulates the original object. Copies of sub–environments can show the user everything happening in the original sub–environments. If the user wishes to operate on some objects in an original sub–environment, he or she can jump into that sub–environment and jump back after the work is done.

1.3.5 Better Response Time

When a user interacts with a virtual environment, instantaneous and continuous feedback is expected. The response time can be measured by the system lag and update rate. If the system lag is too great, the user will get feedback that was expected earlier. If the update rate is too low, the display presented to the user will jump from one picture to the next, rather than appearing continuous. The time for input processing, behavior computation, and rendering all contribute to the slow response time of a VR system. Since sub–environment boundaries can separate different sub–environments, most parts of the virtual environment will not be visible to the user. Therefore, only the current sub–environment, along with those sub–environments which have an open door to the current sub–environment, are possibly visible to the user at any moment. By using the above observation, I am able to reduce input processing time, object behavior computation time, and display time.

Consequently, I can reduce the system lag and increase the update rate so as to improve the response time. Most of the object behaviors can be suspended when the user is not present. Some object behaviors, such as simulations, may not be interrupted, even when the object is not visible to the user. These behaviors are to be kept alive all the time.

1.3.6 Multi–User VR Systems

For any multi–user VR system, either a centralized architecture or a replicated architecture can be used. In a centralized architecture, there is only one copy of the application. The input from each user is sent to the application, and the output is broadcast to every user from the application. In a replicated architecture, each user has a copy of the application. The output to each user is obtained from the local copy of the application. So the update rate of the application is not affected by network delay. Only the input has to be sent to other users. Therefore, the network traffic is much lower than when a centralized architecture is used.

We consider shared users who are at geographically different sites. One of the main problems is to improve response time. The replicated architecture is used to achieve better performance as well as lower network traffic.

Since the replicated architecture is used, each user's local database has to be synchronized to provide all users with the same view of the shared environment. Synchronization mechanisms are provided to help keep the local databases consistent. Users can create objects, remove objects, or modify object attributes. Whenever the shared environment is changed by a user, other users should see the result after the network delay.

When the last user exits from the shared environment, these changes have to be saved somewhere so that the next user who comes into the environment can continue from this point rather than start from scratch. A centralized session manager is provided to manage the changes when no user is present in a shared environment and to coordinate database updates for late-comers. Therefore, this multi–user VR system architecture is actually a hybrid architecture.

1.3.7 Lower Network Traffic

If the users are working in several different sub–environments, changes to some sub–environments may not be visible to all users. We use this fact to further reduce network traffic by not sending changes to those users who can not see the changes at the moment. The changes are only sent to a user when the user moves to the changed sub–environment. If a user never goes into a changed sub–environment, those changes will never be sent.

1.4 Thesis Outline

This thesis has six parts: background, VR environment model, multi–user VR system architecture, multi–user Place system, evaluation, and conclusion.

The second chapter, background, reviews the previous work done in the areas of VR applications, VR devices, and VR software development. The new VR technology can be applied to many areas including visualization, training, computer–aided design, telecommunications, and entertainment. VR devices provide the hardware basis for VR technology. VR input devices obtain gesture, tactile or force, and sound input from users, whereas VR output devices provide visual, haptic, and auditory feedback to users. However, there is little support provided by these devices for programmers to construct VR application systems. The potential of VR applications initiated research on VR software development in the areas of system configuration, single–user and multi–user application architecture, and virtual environment organization. Unfortunately, it is still not easy to build large and complicated virtual environments with reasonable response time and acceptable network traffic using previous VR software systems.

In the third chapter, an hierarchical model for application objects and VR environments is proposed. This model helps VR programmers decompose application design and implementation from a high level to a more detailed level. Application objects can be organized into different sub-environments to improve response time, because only visible objects are evaluated and presented to the local user.

The fourth chapter discusses a multi-user architecture for the proposed model. The hybrid architecture is used for managing VR sessions. A single session manager coordinates user entrances and departures from shared sub-environments. Each user has a Communication Component to communicate with the session manager and other users. Network traffic is reduced since local changes are sent only to the users who can see these changes. The remaining parts of an application use the improved Decoupled Simulation Model. The Geometric Model Component uses the proposed hierarchical environment model. The Interaction Component is responsible for workspace transformation and computation scheduling. Navigation facilities are provided for structured VR environments. The Presentation Component displays only those objects which are possibly visible to the user at the moment.

The fifth chapter presents the multi-user Place system, which is an implementation of the virtual environment model and its multi-user system architecture. This system is written in the object-oriented language C++. Implementation details are discussed in the chapter and an analogy is made between file systems and the proposed virtual environment model.

The Multi-User Place system is evaluated in chapter 6 by doing a theoretical analysis and implementing an example application of the Multi-User Place system.

The last chapter contains a summary and discussion of future work.

Chapter 2

Background

Virtual Reality is getting more attention from the media, public, and industry due to its interesting and promising potential applications. VR hardware devices provide a foundation for exploration of VR applications. However the software support for these devices is minimum. Previous VR software supports the construction of VR applications to some extent, but high-level support for building large, complicated multi-user virtual environments with satisfying performance has not yet been provided.

The research work that is presented here has been done in VR applications, VR devices, VR software development and related areas.

2.1 VR Applications

Virtual reality technology can be applied to a variety of areas such as visualization, training, computer-aided design, telecommunications, and entertainment. Many VR researchers have been working on application systems in these areas.

2.1.1 Visualization

UNC's molecular docking is a typical VR application in scientific visualization. Feiner's "worlds within worlds", and Mackinlay's Perspective Wall explore some techniques to visualize information in 3D environments.

The UNC's molecular docking system was developed to help chemists interactively solve the molecular docking problem by using VR visual and haptic displays [30]. The molecular docking problem finds the right position and orientation for a drug with respect to a receptor site so as to minimize the overall interaction energy. In this system, the visual display presents the positions of both drug and the receptor site. The haptic display imposes the force between the drug and the receptor on the chemist's hand. The chemist can move the drug while feeling the force to try to dock the drug. It was claimed that the chemist obtains a better understanding of the problem by using this system.

Feiner proposed a "worlds within worlds" metaphor for visualizing n–dimensional information in a three–dimensional world, and used the metaphor in a "financial visualization" application [11, 12]. The "financial visualization" system allows the user to examine the values of positions to buy or sell foreign currency on a specified date at a specified price. The values are determined by a function of six variables. In this system, three out of six variables can be kept constant, thus an inner 3D world within the outer 3D coordinate system is obtained. Then general interaction techniques can be used in the inner 3D worlds. If the user assigns another set of values to the inner three variables, a new inner 3D world is produced. By using the "worlds within worlds" metaphor, high dimensional abstract data can be put into three–dimensional space. It is more intuitive and the data is more understandable.

Feiner was using 3D worlds to visualize higher dimensional information. In contrast, Mackinlay designed a Perspective Wall for visualizing 2D information with wide aspect ratios, such as linear information along a time axis [26]. The user's current interest is displayed with details on the center panel while the context is shown on two perspective panels. In this way, the user can focus on the full details of the center panel as well as get context from the perspective panels.

2.1.2 Training

VR can be used in training processes, especially in cases where training on the real equipment is very expensive, such as pilot training.

Kaye et. al. constructed a virtual cockpit system for military pilot training [20]. The system gives the pilot visual and auditory feedback from the virtual cockpit and surrounding environment, which makes the pilot feel that he is flying in a real cockpit.

One of the problems encountered in flight simulation systems is providing information to help the pilot navigate in an unfamiliar large–scale environment. There are two types of maps which are helpful to the pilots. One is the standard north–up type of map. The up direction on a north–up map is the north direction on earth. Another type is the track–up map, which means the up direction on the map is the direction the pilot is facing. The track–up map is ideal for pilot navigation in a large environment whereas the north–up map is used by pilots to communicate with a navigator on the ground. Both track–up map and north–up map are considered to be important in navigation [2, 18].

2.1.3 Computer–Aided Design

UNC's architectural walkthrough and radiation treatment design are two application systems in the area of computer aided design.

The objective of the walkthrough project is to provide the architect with a tool to pre–examine the building he or she is designing, before the building is constructed. It also makes it possible for the clients to "walk through" the virtual building and give some feedback to the architect before it is too late. With this system, the visualization becomes an important part of the architecture design process.

The system used for radiation treatment design is called the virtual simulator [29]. The virtual simulator creates a 3D geometrical patient model which the radiotherapist can accurately and intuitively manipulate. In addition, it provides the radiotherapist with computer–aided design tools to position the radiation treatment beam on the model and adjust the angles so as to maximize the radiation dose to the tumor area, and minimize the dose to nearby healthy tissue. The system helps the radiation therapy physicians obtain optimal designs that will result in higher cancer cure rates with fewer side effects.

2.1.4 Telecommunications

VR technology is believed to be very useful in telecommunications.

Bolas et. al. at NASA Ames Research Center built a telepresence application system. The interface for a telerobotics application not only needs a supervisory control mode, but also an interactive control mode especially when performance degrades. Bolas's telepresence system presents visual, auditory, and tactile feedback from the remote task site. The operator can use gesture, voice, and tactile input to control the remote robot [13, 3]. Thus the operator working in a virtual environment can produce the same effect as if he or she were working in that remote site.

Teleconferences can be held today with telephone or television technology. But virtual reality can be used to go one step further. People in different places can enter one virtual environment and hold a meeting just like they are really in the same room.

2.1.5 Entertainment

Entertainment is a big potential application area of VR techniques. Many video game scenarios can be moved into virtual environments. Because of its sense of reality, it would be more exciting than current video games. Krueger's Videoplace was implemented without goggles and gloves [22] [23]. People can go swimming and skiing inside the Videoplace. The future of VR in entertainment is very promising.

2.1.6 Discussion

The application systems we have described are only explorations of the applications of VR technology. However, we can see that a VR application environment can be really large and complicated like in the molecular docking system, the virtual cockpit system and the architecture walkthrough system. In these cases, the response time problem may arise and the applications may become unresponsive. Furthermore, the above applications are single–user systems, while some of them would require the cooperation of multiple users in reality.

2.2 VR Devices

VR input and output devices are used to provide two–way information exchange between end–users and the computer. The output devices provide visual feedback, three–dimensional auditory feedback, and tactile or force–feedback to the user. Input devices, such as a DataGlove or a speech recognition system, are used by the user to dynamically interact with the virtual environment. Researchers at NASA, UNC, and other institutes have made great progress in constructing and improving several types of VR devices. The work of these groups provides the fundamental basis for research on VR applications and software development.

2.2.1 Criteria for VR Devices

As James J. Gibson stated, the natural environment is the stimulation resource for the human's five senses [15]. VR devices are used to stimulate the human body's visual sense, auditory sense, and tactile sense. The devices dealing with vision, tactile sensation, and the sense of hearing have been studied. Stereoscopic display systems are used to show three–dimensional pictures to the user's eyes. 3D sound systems can synthesize and play three–dimensional sound in the virtual environment. Force–feedback systems put specified amounts of force on the human body. Devices to stimulate the smell and taste senses might be invented in the future, but no immediate application acts as a strong motivation to initiate research in these areas. In addition to devices that provide environment information to the user, interaction devices have also been invented for the user to tell the computer what he or she would like to do.

VR devices are fundamental parts of VR systems. In some sense, VR devices are used to fool the human's senses in order to replace the natural environment with the virtual environment. It is a tough job, to fool the human senses, since they have been evolving for so many years. The VR devices must be reasonably satisfying so the end user can bear to live within the environment, at least for a while.

To generate a reasonably realistic environment, these devices must meet minimum requirements in terms of realism, response time, and usability. Realism means that there is not excessive distortion in the images a sensory channel presents, and that the noise level is low enough. For example, in the case of visual feedback, the picture shown to the user must be drawn without any distortion and jitter. Response time refers to the system lag and update rate. If the system lag is too big, the end user will see the picture he or she should have seen a moment ago. While, if the update rate is not fast enough, the display will jump from one picture to the next instead of being continuous. Usability is also an important requirement for VR interaction devices, since the VR environment is a type of graphical user interface.

In this section, VR devices will be examined from the following perspectives: hardware configuration, software support, and system performance including realism, response time, and usability.

2.2.2 Visual Display Devices

Hardware configuration

Basically, there are four types of VR hardware configurations for generating stereoscopic images.

The first stereoscopic display system was introduced by Sutherland in 1968 [38]. The idea was to surround the user with a three–dimensional environment. The display system presented three–dimensional illusions to the user. When the user moves his or her head, the images change in the same way as when the user is moving around in a real environment. A mechanical or ultrasonic head position sensor was used to detect the position and orientation of the user's head. Then the corresponding perspective image was computed and sent to the special spectacles with two miniature cathode ray tubes (CRT) for display. The spectacles used half–silvered mirrors in a prism, so the user could see the picture from the cathode ray tubes as well as objects in the real room. This is a kind of see–through head–mounted display (HMD).

The virtual environment display system built by NASA contained a non–see–through stereoscopic head–mounted display [13]. Two medium–resolution, monochromatic liquid crystal display (LCD) screens using standard NTSC signals were used for display. The LCD technology was used to get a small, light, safe, power–saving and low cost display system. A 6 degree–of–freedom tracking device mounted on the helmet was responsible for tracking head movements. The VPL EyePhone is the same kind of stereoscopic head–mounted display [41].

Another system also built at the NASA Ames Research Center is a counterbalanced CRT–based stereoscopic viewer (CCSV) [27]. It was developed to provide higher resolution than a LCD based head–mounted display could provide. The CCSV was mounted on a 6 degree–of–freedom counterbalanced arm. The user can use the handles on the side of the display to move the CCSV to track head motion. The position and orientation data of the display head can be determined from transducers at the joints of the arm. The CCSV is not head–mounted so the user can easily go into the virtual world, look around and come back to his or her desk top environment.

The last type of stereoscopic display is totally different from the first three in that it does not require the user to either wear special glasses or move the equipment to follow him or her. The principle behind this direct–view type of stereoscopic display system is to display the image on a 7–inch LCD which is covered by a cylindrical lens array, called lenticular lenses [1]. The lenticular lenses divide the left and right images and presents them to the user's eyes. A highly directional infrared transmitter and position sensing device (PSD), mounted on both sides of the LCD display, is used to detect the head position.

Software support

In order to provide a better device level interface to the VR application programmer, some software packages have been developed on top of the stereoscopic display devices.

In Sutherland's system, there was a matrix multiplier. Every time the position and orientation data arrived, the matrix multiplier would transform the information from the room coordinates to eye coordinates to produce the perspective view for the user. So the application program did not have to repeat this routine work.

Similar to Sutherland's matrix multiplier, CCSV has Forward Kinematics. Forward Kinematics converts the joint angle information to the display head's position and orientation, which is used to set the viewing parameters in the applications.

Here at the University of Alberta, our graphics group developed the Minimal Reality (MR) Toolkit to help with VR application construction [36]. The MR Toolkit uses the VPL EyePhone for display and it has device level software dealing with 3D devices. The software packages in the MR Toolkit provide higher level interfaces for the devices. One of them is the workspace mapping package. Since the VPL EyePhone has its own coordinate system that depends upon its location in the room, the workspace mapping package provides routines to convert the device data from the device coordinate system to the room coordinate system, then to the application's virtual world coordinate system. In addition, since we use two graphics workstations to produce the images, there are facilities to synchronize the display of the pictures for the two eyes.

System performance

As mentioned before, system performance for VR devices involves realism, response time, and usability.

In the case of stereoscopic display devices, what we mean by realism is that the images presented to the user should be of high quality without any distortion and without jitter. Resolution is the first issue. CRTs can definitely provide higher resolution than LCDs can. But when considering size, weight, power, safety and cost, the CRT is probably not the better choice. The second issue is picture distortion. In Sutherland's system, both the cathode ray tubes and the head-mounted optical system contribute to the so-called pin-cushion distortion which was about three percent [38]. The NASA HMD, the CCSV, and the VPL EyePhone all use the Large Expanse Extra Perspective (LEEP) optics designed by Eric Howlett [27, 47]. These optics provide a 120 degree field of view but introduce chromatic aberrations and pincushion radial distortion. Chromatic aberrations usually occurs along the edges of the images with highly contrasted color. The effect of pincushion distortion is not obvious so this problem is often ignored. Robinett and Rolland at UNC proposed a computational model for the geometry of a HMD [35]. This model considered the physical geometry of all device components that may affect image quality. It is claimed that this model compensates for all the distortions caused by the display screens, the optics and the placement of the screen with respect to the eyes. The last issue is the jitter problem. Jiandong Liang at the University of Alberta used an anisotropic low pass filter to reduce the noise in head position data in order to solve the jitter problem [25].

Response time involves minimum system lag and high update rate. Bryson at NASA Ames described how to measure the system lag in [4]. Rebo and his colleagues at Air Force Institute of Technology used a simple Kalman filter to do predictive tracking of the user's head position to reduce the system lag [33]. Based on the observation that the delay in orientation data is the major factor causing the lag, Jiandong Liang designed another predictive Kalman filter to compensate for that delay so as to minimize the system lag [25]. The update rate is actually a measure of image continuity. Generally speaking, it should be thirty frames per second for human users to see smooth motion, but as few as ten frames per second can be acceptable. The update rate of a computer system depends upon the CPU speed of the host machine, as well as how much device management, application computation, and drawing has to be done per frame. One way to get higher update rate at the device level is to distribute the workload to several different machines as is done in the MR Toolkit of University of Alberta [36].

Usability is based on how comfortable and convenient the user feels using various stereoscopic display devices. The head–mounted display gives the user a feeling that he or she is in a three–dimensional environment. But the user's eye will get tired after a while due to the LCD screens' low resolution. The fact that HMDs are quite heavy could cause neck fatigue. In addition, the cables of HMDs may restrict the user's movement. CRT–based stereoscopic display systems have higher resolution, but are much heavier than LCD displays. So they are usually not head–mounted. As in the CCSV system, the display head is mounted on a counterbalanced arm. The user has to move the display head using his or her hand, which is not as convenient and natural as in a HMD system. Also the user's movement is confined to the positions that the arm can reach. The direct–view stereoscopic display system does not require the user to wear any kind of special glasses and still can present stereoscopic images to the user. However, the user doesn't have the feeling of being within a surrounding environment because the display screen is not large enough. Another advantage of the CCSV system and the direct–view stereoscopic display is that the user is not completely separated from the real world as one is when wearing a HMD. In this case the user is able to use traditional devices, such as keyboard or mouse, as well as 3D VR devices to interact with the VR environment.

2.2.3 Auditory Display Devices

Besides the visual devices, auditory devices provide another channel for the user to get information from the environment. As Wenzel et. al. stated, three–dimensional sound information can greatly improve the user's understanding of the environment [44]. Although auditory devices are useful in VR systems, only a few VR researchers have done work in this area. Because scientists do not fully understand how the humans' auditory system works, there is still a lot that can be done as more psychological research results come out.

Hardware configuration

In order to produce three–dimensional sound, Doll et. al. used many real sound sources or loudspeakers [10]. But in virtual reality, the virtual sound sources might be dynamically specified anywhere within the environment. Wenzel designed an auditory display device to process the sound signal to synthesize three–dimensional sound cues in real–time and present them over headphones to the user [44]. The psychological theory behind the synthesis technique is that human sound localization is not only determined by the "interaural differences in time of arrival at low frequencies and interaural differences in intensity at high frequencies", but also determined by the direction–dependent interaction between the incoming sound waves and the outer ears or pinnae, as cited in [45]. The acoustic adjustment of the sound signal by the outer ears is defined as the Head–Related Transfer Function (HRTF). To sample the HRTF, Finite Impulse Responses (FIRs) are obtained through experiments on a specific subject. Then the listener–specific "location filters" can be built and used in producing three–dimensional sound signals [44]. The sound can be simulated to be anywhere outside the headphone in the environment by interpolating the signal with linear weighting functions. Since it is impossible to measure and store the FIRS information for every potential listener in practice, Wenzel et. al. have gone one step further to study the feasibility of acoustic devices based on non–individualized virtual acoustic sound cues [45].

Software support

In Wenzel's paper and other published papers about three–dimensional auditory display devices, software support is not discussed. However, it would be very helpful to have some software packages to provide high level facilities to the programmer, such as a package to facilitate the construction of a sentence — a sequence of different sonic frequencies.

System performance

The realism standard for acoustic display devices is that they should produce simulated sound as if the sound were produced by a free-field source. Psychological research has been done on comparing free-field and synthesized free-field listening [43]. The results show that the use of the synthesis technique is satisfactory for static sources when using listener-specific "location filters". However, confusion errors remain quite high when non-individual HRTFs are used [45] .

The real-time performance of this device is acceptable according to Perrott's work on the perception of auditory motion [32]. The digital signal-processor has a maximum lag or directional update interval of 10–30 msec. There may be additional latencies added by the other parts of the system.

As to usability, this acoustic display system requires that the user wears headphones. Nevertheless it provides static or moving virtual sound sources, and the sound can respond to head movement when coupled with head-tracking devices.

2.2.4 Haptic Display Devices

In addition to the visual and auditory display, haptic display is another channel to deliver environment information to the user. Force-feedback has been studied in the robotics field for applications in telerobotics. Now researchers at UNC, the University of Tsukuba in Japan, and MIT are working on various haptic display devices that can be used in VR applications including telerobotics.

Hardware configuration

UNC's force display system uses a molecular docking problem as the driving application. In this system the user can feel the interaction force field between a drug and a receptor site in a protein or nuleic acid [30]. An Argonne E-3 Remote Manipulator (ARM) is used as the force display. When the interaction forces and torques among the molecules are computed, the manipulator will impose the forces on the user's hand as if the user were holding the drug. The user can also use the handgrip of the manipulator as a 6–D joystick to dock the drug into the receptor. So the manipulator is a force output device as well as a 6–D input device.

Similar to UNC's system, the force–feedback device designed by Iwata at the University of Tsukuba in Japan is a "tactile input device with reaction force generator" [19]. The manipulator is used to measure the joint angles of the user's hand as well as impose forces on the fingers and palm of the user.

Patrick et. al. at MIT designed a non–reactive tactile display device [31]. As they state, a reactive display exerts joint torques on the operator's body, thus stimulating the body's cutaneous and musculo–skeletal sensors; while a non–reactive tactile display does not exert torques to the joints and stimulates only the cutaneous sensors. Their non–reactive tactile display has the advantages of being light in weight, small size and low cost. Two voice–coils (small speakers), which are used as the tactile display, are mounted on the fingertips of one's thumb and index finger. They are driven by a 250 Hz variable–amplitude signal from analog electronics units. A PC samples the hand's position from sensors on the EXOS Dextrous Hand Master (DHM) and uses this data to control the electronics units.

Software support

Haptic device studies are still in the very early stage. We know little about how to precisely deliver a specific feeling to the human body through these devices. There are no software packages that can be used to produce special kinds of tactile or force feedback to the user. Minsky et. al. at UNC has started to do experiments on texture feeling [28]. In their Sandpaper system, they use tiny virtual springs to pull in or push away the user's hand from a surface to make the hand feel the textures. Experiments like the Sandpaper system will greatly help the development of haptic software tools.

System performance

Realism, response time, and usability are still three important issues with haptic display devices.

None of the above three devices have reached the point where they produce realistic haptic displays. For example, a virtual wall produced by MIT's non–reactive tactile feedback device feels a little bit like a vibrating plate [31]. The high friction associated with the cable–driven system is a big problem for UNC's force display system [30]. Iwata used an intuitive idea to present slightly more realistic force–feedback. When the user's hand grabs a rigid object, the maximum torque was imposed on the fingers to tell the user that he is grabbing a hard surfaced object. Whereas the force imposed on the palm depends upon the relative position of the palm and the object, as well as the mass distribution of the object [19].

Response time is not quite satisfactory either. As Ouh–young et.al. stated in [30], at least 20 updates per second are needed. For UNC's system, the latency between the detection of a hard surface and when the servo motor is started is 200–300 msec. And the overall system update rate is about 3–5 updates per second. For the solid model virtual space example, Iwata's system has an update rate of 4 Hz, and a lag time of 0.25 seconds. MIT's non–tactile force feedback system claims to have 60 updates per second. That is probably because the device was so easy to drive and the application was having the user find the position of a virtual wall.

Just as the user has to wear an EyePhone or headphone to get three–dimensional visual and auditory feedback, the user has to put his/her hand in the special haptic devices to feel the tactile or force display. It may not be comfortable for the user. Even worse is that it could be very dangerous. The UNC system uses two measures to guarantee the safety of the user [30]. One is to use a dead–man footswitch. The user must step on it to have power on the servo motors. The second is to use the safety–range detector to restrict the maximum force the manipulator can put on the user. Other issues about the usability of haptic display devices, such as allowing the user to move around while receiving the force feedback, have to be further studied.

2.2.5 Interaction Devices

By interaction devices we mean VR input devices. All the above devices are essentially output devices for VR systems, the end-user needs various input devices to interact with the VR system. These input devices are used to visually, aurally, and tactually represent the user within the virtual environment, so that the user can influence the environment through these devices.

Hardware configuration

The DataGlove is a device used to visually represent the user's hand in a VR environment. It has two parts: flex sensors, and a Polhemus sensor [40]. The flex sensors refer to the ten optical fibers and a optical adaptor that has the light sources and phototransistors. The optical fibers are on the back of a common glove, along the joints of the thumb and fingers. When the optical fibers are bent, the amount of light going through them is reduced. The reduced light level is converted to an analog voltage by a phototransistor and sent to a control unit. The Polhemus sensor used to track the position of the DataGlove is the same as the one used to track the head position in VPL EyePhone. It contains two components: a source and a sensor. The source emits the electromagnetic fields. Within this field, the sensor can tell its position relative to the source and produce analog signals describing the position and orientation of the DataGlove to the control unit. The control unit converts the analog signal from the DataGlove to a proper data format for the host computer. This data can be used to draw the hand of the user on the screen.

Speech recognition techniques (used in NASA's virtual environment system, see [13]) can be used to input user commands. But it is not a typical VR device. Speech recognition can be used in various areas beside virtual reality.

The user can also influence the virtual world tactually. As we discussed before, the tactile or force display devices are also tactile input devices. When the user moves his/her fingers to impose force on the devices, the devices detect the effect and transmit the data back to the VR system.

Software support

VPL has developed the "DATAGLOVE GESTURE EDITOR SOFTWARE For the Apple Macintosh" [39]. Here at the University of Alberta, we have a DataGlove calibration and gesture editing program that can interactively define the static gestures as well as recognize gestures at run time [36].

System performance

We have discussed the system performance of tactile input devices.

For the DataGlove, the noise and response time issues have not been thoroughly studied. Filters for reducing the noise and delay are still to be developed. VPL once tried to use the DataGlove technique to build a datasuit. But the delay problem made it impossible to get reasonable response times.

Usability is the big issue here. The DataGlove is not uncomfortable to wear. However, it is not the best choice. Due to the noise problem, it is not easy for the user to do a selection or pointing operation, or to make the exact gesture required to issue a command. Even if we could solve this problem, the potential limit on the size of gesture vocabularies would restrict the number of commands.

2.2.6 Discussion

From the survey of various VR devices, we can see that VR device research has a long way to go. First of all, all the current devices need to be improved in hardware. Maybe new devices will need to be designed. Second, there are little software support available for the VR programmer to get better service from some of the devices, probably because the hardware problems have been keeping device researchers occupied. Third, realism and response time must be greatly improved. And finally, the devices must be refined so that users feel comfortable using them.

However, research in VR devices did provide a reasonable basis for research on upper level software and application systems. It also resulted in the general requirements of VR systems such as realism, response time, and usability. Realism and response time are not only goals for devices, but also for VR support software and application systems, so as to produce a realistic, dynamically interactive environment. To reach such an objective, both realism and response time are important. Unfortunately, improving realism takes time and consequently affects the response time. As a result, we have to make trade-offs between the two in order to improve the realism while keeping the response time acceptable. As to usability, we have discussed the issue mostly from the physical perspective — whether the user is physically comfortable with the devices. When we build software facilities and VR applications, we have to consider the issue from the psychological point of view, since virtual reality is supposed to be a more user-friendly type of user interface.

2.3 VR Software Development

Based on the experience gained from previous VR application systems, people began to work on software tools for supporting the development of future application programs. In this section, we will review some research work related to system configuration, single-user and multi-user application program structure, concurrency control methods, and VR environment organization.

2.3.1 System Configuration

System configuration is the most fundamental part in constructing any VR application. It is responsible for the management of all VR devices, providing effective and convenient services.

Since VR device drivers take a lot of run time, the application task and related software will not get enough CPU time to have the high performance that VR systems require. Researchers at IBM use multiple workstations to distribute the workload and an event-driven device management system to coordinate these VR devices [42, 24].

The Minimal Reality (MR) Toolkit developed here at the University of Alberta uses a similar approach to distributing the device drivers [36].

2.3.2 Single–User Systems

At the top level of the whole system, there are application programs. Usually, an application addresses one specific task. One application can be a single–user application or shared by several users. Several research groups have suggested different structures for single–user VR application programs.

Zeltzer and his colleagues at MIT produced a general–purpose package for building interactive simulation systems, especially for task–level animation systems [46]. The key element is a constraint network which connects all the objects. Once the status of an object is updated, all the constraints which involve that object are informed and evaluated. This process continues until no more objects are changed and then the final result is displayed.

Robertson et. al. at Xerox proposed an architectural model called the Cognitive Coprocessor Architecture [34]. The purpose of the Cognitive Coprocessor Architecture is to support "multiple, asynchronous, interactive agents" and smooth animation. It is based on a three agent model of an interactive system. These agents are: the user, the user discourse machine, and the task machine. The basic control mechanism is the animation loop, which has a task queue, a display queue, and a governor. The task queue maintains all the incoming computations from different sorts of agents; the display queue contains all the objects to be drawn; while the governor keeps track of time and helps the application to produce smooth output.

At the University of Alberta, the Decoupled Simulation Model is proposed in order to improve the response time of VR application systems [36]. In the Decoupled Simulation Model, there are one or more UNIX–style processes in an application. One is the master process; others are slave or computation processes. The master process establishes socket connections with server processes and gets input from the user. To reduce the communication delay between processes, the master process also displays the output to one of the user's eyes. Slave processes are responsible for output. The output to another eye of the user is produced by a slave process of an MR application. Computation processes perform the simulation task. Since the computation processes run asynchronously from other processes, the whole application system is sped up. That is why the model is named the Decoupled Simulation Model.

2.3.3 Multi–User Systems

The above three models are suitable for single–user applications. When we consider multi–user applications, we have to decide whether to use the centralized architecture or the replicated architecture.

Centralized Architecture

In a centralized architecture, there is only one copy of the application. The input from every user is sent to the application and the output is broadcast to every user from the application.

IBM researchers used the centralized architecture to build a multi–person virtual world called rubber rocks [8]. Two users can manipulate several flexible objects simultaneously. For each user there is a Dialogue Manager which manages input from input devices, sending output to output devices, and communicating with the rubber rocks simulation. Thus each user may have his or her own interaction interface. But their Dialogue Managers talk to the same underlying simulation.

Replicated Architecture

In a replicated architecture, there is one copy of the application for each user in a multi–user environment. The input from every user is distributed to every copy. The output to users is obtained from the local copy of the application. Somehow all copies of the application must be synchronized and the output to users should be consistent all the time.

People in the field of computer–supported cooperative work use the replicated architecture for computer teleconferencing. GroupSketch designed by Greenberg is a sketchpad for multi–users who are geographically distributed at different sites [16]. It allows up to four users to draw, enter text or gestures on the shared work surface on a virtual piece of paper. Any user can join the collaboration after others have started and leave the collaboration as he or she wishes. GroupSketch uses the replicated architecture. On each workstation, there is a copy of the application and a conference agent. A registrar daemon is responsible for introducing a late comer into a GroupSketch session. Each copy of the application communicates with the registrar through its conference agent. Applications are synchronized by sending events to each other.

The SIMulation NETworking (SIMNET) system [5] developed by LORAL Advanced Distributed Simulation Inc is probably the first virtual reality system which uses the replicated architecture. The SIMNET system is aimed at team training of many soldiers in a virtual battlefield. The battlefield is composed of static components and dynamic components. The static components are the "passive elements" in the virtual world while the dynamic components are the "active participants" that are "moving and changeable objects" in the environment. Every user has the same static components and the change–of–state events of the dynamic components are broadcast to all users to ensure a consistent view for each user. The event receivers are responsible for deciding which events are of interest to themselves and what effect an event may have. As a result, any user can join or leave the shared environment at any time. In order to save network bandwidth, the event receivers can use the "dead reckoning" technique to extrapolate states of those dynamic components so that the change–of–state events do not have to be sent out continuously.

The Distributed Interactive Virtual Environment (DIVE) system [7] built at the Swedish Institute of Computer Science also multicasts changes to the replicated databases residing in all the participating processes. The database is composed of completely different worlds, each of which contains a specific set of objects. The worlds are implemented by process groups, where one DIVE process represents a user or an application. A DIVE process belongs to one world at a time and it can switch worlds dynamically which means a complete switch of context.

Similar to the process group concept, The WAVES (WAterloo Virtual Environment System) architecture [21] designed by Kazman at University of Waterloo uses a host to simulate a subset of objects, thus a collection of hosts is required to create an entire simulated world. Some of the objects are native to a particular host while others can be clones of objects which are native to other hosts so that users associated with different hosts can share these objects. The message managers mediates communication between hosts but hosts can not communicate directly with each other. In order to minimize message volume, the message managers can filter the messages that a particular host receives upon request. But this feature had not been implemented when the paper was published. The "dead reckoning" technique was also extended to predict objects' behavior so as to reduce network traffic.

Similar to WAVES, the BrickNet system [37] developed at the National University of Singapore has clients with different virtual world contents. A client can deposit its objects at a server while other clients can lease objects from any of the servers on the network. A user can use "dynamic portals" to go to another world, then select and copy objects from that world to the local world. Updates on the shared objects are sent only to the clients who have the objects and are interested in the updating of the objects. BrickNet is aimed at providing a "network–based design environments where a complete design task is distributed over several design nodes" and each node, a client, "shares its design objects with others to facilitate collaboration on the larger design task." However, it can also be used to build applications with the same content for every user. For example, a ship planning environment was built to allow several users to "load containers destined for different parts of the world in a ship." In this case, all changes will be sent to every user in the shared environment.

The replicated architecture is more complicated than the centralized architecture. The advantage of the centralized architecture is that it is simple. The application program does not have to be changed. There is no need to keep different copies of the application consistent. As stated in [9], the replicated architecture has the following advantages: better performance and easy site–specific interaction. Because only the input from users has to be distributed to every copy of the application, network traffic is greatly reduced. Every site has the local copy of the application, so output is less sensitive to network latency. Therefore, better performance can be achieved. In order to get site–specific interaction, like in IBM's rubber rocks simulation, applications using the centralized architecture must keep separate event–processing threads for each user. Replicated applications are single–threaded which makes site–specific interaction easy. These advantages of the replicated architecture are very important for VR applications.

2.3.4 Concurrency Control

When a replicated architecture is used, the application copies at different users' sites have to be synchronized by sending messages to each other. However, inconsistencies will arise when two or more users try to change the same object at the same time. At each user's site, the remote updating message will arrive after the local update. Consequently, the object will be updated in different orders at different sites. A concurrency control mechanism must be employed to keep the application in a consistent state.

Traditional Concurrency Control Methods

Concurrency control maintains application consistency through serialization which ensures that the result of interfering events is equivalent to the result of these events being executed in one order at all sites. In [17], Greenberg and Marwood reviewed the traditional concurrency control methods and discussed the problems that occur when these methods are directly applied to real time distributed groupware systems.

Serialization techniques can be non–optimistic or optimistic.

Non–optimistic serialization does not allow events to be executed out of order. It is guaranteed that these events are executed in the same order at all sites. While one event is changing an object, the second event, which is trying to change the same object, will not be executed until the first one is finished. Even when an object is not modified by any event, the application still has to check before an event is allowed to start changing the object. As a result, the response time of the application is very slow. In a groupware system, this "interferes with the flow of the interaction and begs the question of how feedback to this 'pending' state should be handled." [17]

Optimistic serialization allows events to be executed out of order. It then detects and repairs the results so that the final results appears to be the same as if the events were executed in a correct order. However, the implementation of detecting and repairing out of order events could be quite expensive. If a groupware system uses this approach, a user who is trying to manipulate an object that is being updated by another user could see the result of his/her own action first, then suddenly see the result of the other user's action. This type of situation will make groupware users very confused, as stated in [17].

Locking is a widely used serialization technique. An application may have locks on different levels of objects. Locking controls the order of event execution by making the events obtain and release the locks on the objects they operate on. The granularity of the locks determines the degree of concurrency of the application. The coarser the locks, the lower degree of concurrency and "the more difficult it is for people to work close together." [17]

A locking scheme can be non–optimistic, semi–optimistic or fully–optimistic.

Non–optimistic locking does not allow an event to be executed before the lock is granted. As discussed in non–optimistic serialization, it results in an unresponsive interface. In addition, the application has to show that an object is waiting for a lock.

Semi–optimistic locking allows the event to manipulate the object but blocks the event at the end of its execution. In case the lock is denied, only this event has to be rolled back, which should be easy for users to understand. However, if the granularity of the lock is finer than necessary, semi–optimistic locking could produce a jerky interface just as non–optimistic locking.

Fully–optimistic locking not only allows the event that is applying for the lock to be executed, but also allows the application to move onto the next event before the lock is granted. It avoids delay and produces a responsive interface. However, if the lock is denied, the whole chain of events have to be rolled back. If a user had started to work on something else, he or she may not notice the reversion. Furthermore, it is not obvious to the users whether the reversion is due to the undo action or another user has changed the object.

The concurrency control methods have different impacts on an application's interface and consequently affect the end users. As claimed in [17], the user–mediating policy would be enough for a groupware system most of the time because groupware users usually "follow social protocols for mediating interactions". The concurrency control methods can be used in rare cases where conflicts do occur. The results of these methods must match what the users expect, otherwise they will cause confusion.

Greenberg and Marwood pointed out in [17] that none of the aforementioned methods are ideal for all real time distributed groupware systems because of user involvement. They designed a concurrency scheme library from which an application can choose the most appropriate one according to the specific requirements of the application.

Concurrency Control in multi–user VR systems

The SIMNET and WAVES systems did not consider the concurrency control problem. Actually, in the SIMNET system, each dynamic object is responsible for updating its own status and broadcasting updates to other sites. There are no cases when more than one user can change an object at the same time. Therefore, the status of every dynamic object will always be consistent across all the sites. That is, the SIMNET system does not have the concurrency control problem.

DIVE and BrickNet used non–optimistic locking schemes to prevent potential conflicts when shared users are manipulating the same object.

DIVE used mutually exclusive locks. Whenever a process asks for a lock on an object, the system will check to see if there is another process holding the lock. If the answer is yes, the requesting process has to wait. Otherwise, it gets the lock. After the process finishes its work, the lock is released and can be used by other processes.

The BrickNet system uses the concept of object ownership. A user has to apply for the object ownership from the server before changing an object. The ownership is given up after the change is made so that somebody else can use the ownership.

2.3.5 VR Environment Organization

Usually people would like to organize their working objects into groups. But none of the above VR systems supports the construction of structured application environments. W. Bricken et. al. at the Human Interface Technology Laboratory at the University of Washington worked on a so called "Virtual Environment Operating System (VEOS)". The basic idea in VEOS is that everything is an Entity. However, VEOS does not provide support for virtual environment organization.

In the Information Visualizer, Card et. al. used the concept of 3D Rooms to divide the information workspace into different Rooms [6]. The current Room stores the most immediately accessible information, while other 3D Rooms store information which might be of interest to the user sometime later. And the user can switch from one Room to another. 3D Rooms provide a nice tool to organize a large information workspace. But a 3D Rooms user can not access information in another Room from his or her current Room. In addition, 3D Rooms was not originally designed for virtual reality systems, and is not a multi–user system either.

2.3.6 Discussion

The VR systems that are reviewed here mainly attacked two problems: the response time problem and the network traffic problem. Nevertheless, none of them considered making use of environment structure to improve response time and to lower network traffic.

WAVES and BrickNet systems allow different users to be in different worlds. If each user has a small local world, the response time will not be a problem because only the local world has to be updated and presented to each user. However, if any of the local worlds contains a large number of complicated application objects, the response time problem still exists.

In order to reduce network traffic, the BrickNet system passes any object changes only to the users who share the object. If two users are to share a large set of objects over a long period of time, while they might not work closely together all the time, it may not be necessary to send all object changes from one to another.

The idea of 3D Rooms can be used to organize large virtual environments into hierarchical structure. Unfortunately, the 3D Rooms system was not designed for the purpose of improving the response time and network traffic of VR application systems. A VR software system is required to provide framework for virtual environment structure and support better response time and lower network traffic based on this environment structure. An example of the potential applications for such a VR software system is a car design environment in which many engineers can work on different parts of a car model in different sub–environments.

The Place system described in the following chapters is implemented to address the above requirement.

The concurrency control problem will not be discussed in this thesis therefore is a limitation of this thesis work.

Chapter 3

Virtual Environment Model

This chapter introduces the hierarchical virtual object model and virtual environment model. The virtual object model is used to build application objects so as to help VR programmers design and implement VR applications from a high level to a more detailed level. The virtual environment model, the Place model, is used to organize application objects into different levels of sub–environments. A user can work on a small number of application objects in one sub–environment at a time. So the response time is improved, since only the objects which are visible to the user are evaluated and displayed to the user. The network traffic of a multi–user VR system is reduced by sending object changes only to the users who can see the changes.

A special type of object—door—may be used to aid users to navigate between sub–environments. If a user would like to view or modify an object which is in another sub–environment, a virtual copy of that object in the current sub–environment will allow the user to perform the task without leaving the current sub–environment. The crystal ball, which is a virtual copy of another sub–environment, enables a user to see what is happening in that sub–environment from the current sub–environment.

An example application will be used to illustrate the models as they are presented. The example environment is a small house sitting on a piece of grassland. In the house, there is a living room, a kitchen and a bedroom. Users can look outside of the house through a window in the living room. There are also a clock, a chair and a sofa in the living room. In the kitchen, two donuts are being cooked in an oven which is on a kitchen table, and there is a fan beside the table. The bedroom has a bed as well as a clock. When the users are in the bedroom, they might also want to know if the donuts are cooked in the kitchen.

This example will also be used in the following two chapters. The example program is given in the evaluation chapter.

3.1 Virtual Object Model

A virtual object (VO) is anything that can be put in a virtual world. The clock, chair and donuts are all virtual objects. A VO has physical properties. It has its own appearance so that users can see what shape it is, what color it is, etc. It may produce sound so users can hear its voice, for instance, a clock may beep at a specified time to remind users of pre–planned activities. When a user tries to touch a VO, it may feel hard or soft. It may even have smell or taste such as a donut. A VO may also have other object–specific properties. Of course the corresponding devices must be available to present these physical properties to the end users.

Some VOs also have behaviors. A VO behavior is any action taken by the VO which may cause a change in the virtual world. There are two types of behaviors. The first type of behavior is reactions to the users' interactions. These behaviors are called event–handling behaviors. The event–handling behavior of the chair reacts when the local user does a grab gesture close enough to the chair's position. When such an event happens, the chair will translate itself, following the hand's movement, to a new position. The second type of behavior does not need any user's interference to activate them. They keep changing objects until terminated. These behaviors are called animation behaviors. The clock's animation behavior is to get the current time from the host computer and report it to the user; while the donuts' animation behavior is to be baked, i.e., to change their colors as time passes. Depending upon the application, some animation behaviors can be suspended when the user can not see the results. Others, including time-consuming simulation computations, may have to be running all the time to avoid side-effects. For example, the clock could stop reporting the current time when it is not visible to the users. However, the donuts should be baked whether the users are present or not.

A VO can be composed of several parts, - called sub-VOs, which in turn are the super-VOs of their sub-VOs. The clock, as an example, has the following sub-VOs: a frame, twelve dial markers, an hour hand, a minute hand, and a second hand, which tell the current time. The typical event-handling behavior and animation behavior of a super-VO is the collection of its sub-VOs' event-handling behaviors and animation behaviors. The animation behavior of the clock is to get the time from the host computer and invoke the animation behaviors of the hour hand, minute hand and second hand to adjust their angles so as to display the time. This hierarchical structure makes it possible to decompose the application system design and implementation into the bottom level object design and implementation.

A VO can have multiple virtual copies in the virtual world. These virtual copies, which are also VOs, have exactly the same attributes as the original VO except for position and orientation. Thus the VO may appear at different locations, at different view points to the user. The clock in the bedroom can be a virtual copy of the clock in the living room. So the user in the bedroom does not have to go to the living room all the time to check the current time. In addition, two VOs could share one sub-VO in their object hierarchy. The original sub-VO has the position and orientation relative to its super-VO, while the sub-VO copy will have its position and orientation relative to the super-VO of that copy. A typical example is a door connecting two neighbor places, as will be discussed in the next section.

Figure 3.1 shows the virtual object model.

One special type of VO is the representation of the user. A user consists of several sub-VOs. Depending on the VR devices locally available, these sub-VOs could be eyes and hands corresponding to the EyePhone and the DataGloves respectively. The position and orientation of the eyes and the hand, plus the hand gesture are

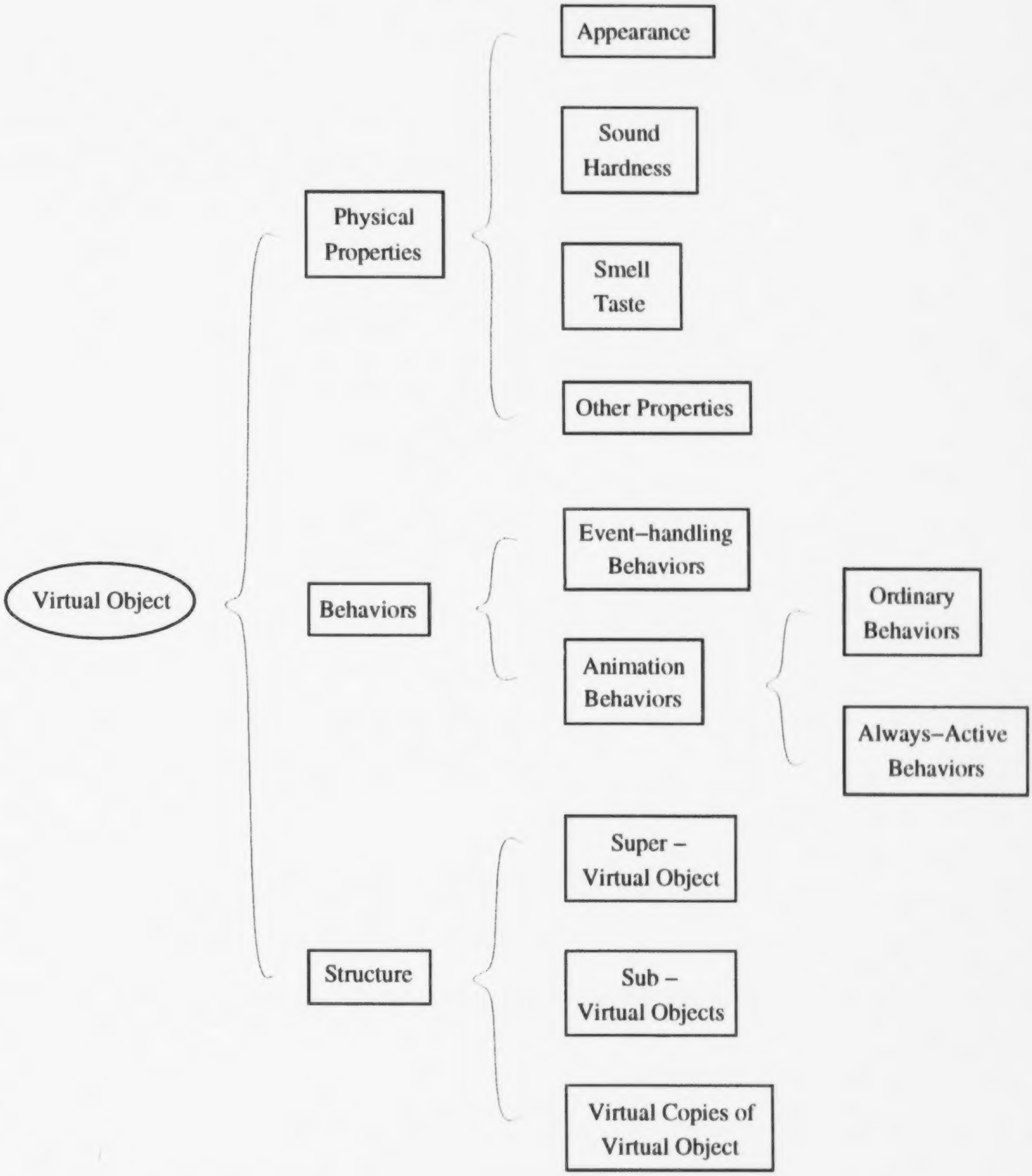

Figure 3.1: The Virtual Object Model

controlled by the real user who is in the virtual world. These inputs are the original event sources, which trigger other event-handling behaviors in the user VO and other VO's. One of the user's behaviors is flying around within the virtual environment without moving in the physical world.

The relationship among VOs is reflected by their behaviors. The behaviors of one VO might change properties of another VO or several other VOs.

3.2 The Place Model

Figure 3.2 shows the Place model. The whole VR environment is commonly called the virtual world (VW). The VW is composed of one or more virtual places. The term "place" is borrowed from real life. In a virtual world, a virtual place has a more limited meaning than a real place. A virtual place (VP) is a space with a definite boundary. In the example environment, the house, kitchen and bedroom, as well as the oven are all VPs. A boundary can be composed of one or more VOs like walls. The boundary could also have behaviors, such as not allowing the user to go out of the boundary. These boundary VOs could be composed of multiple levels of sub-VOs, just like any other VOs. If the boundary blocks the view of the outside of the VP in all directions, this VP is a closed VP. Otherwise, it is an open VP. For instance, the house is an open VP because users can see the outside of the house through the window; while the kitchen and the bedroom are both closed VPs. Closed VPs are very effective in reducing the response time of virtual environments because the application objects that are not in the user's current VP — a closed VP — are not visible to the user any more. Therefore, the user can not interact with these objects which saves the event-handling behavior evaluation time of the objects. Furthermore, there is no need to evaluate the objects' ordinary animation behavior, and to render these invisible objects. Consequently, the overall response time can be greatly improved. Open places, on the other hand, gives VR programmers flexibility to build places like house where an user may want to see the objects outside of the

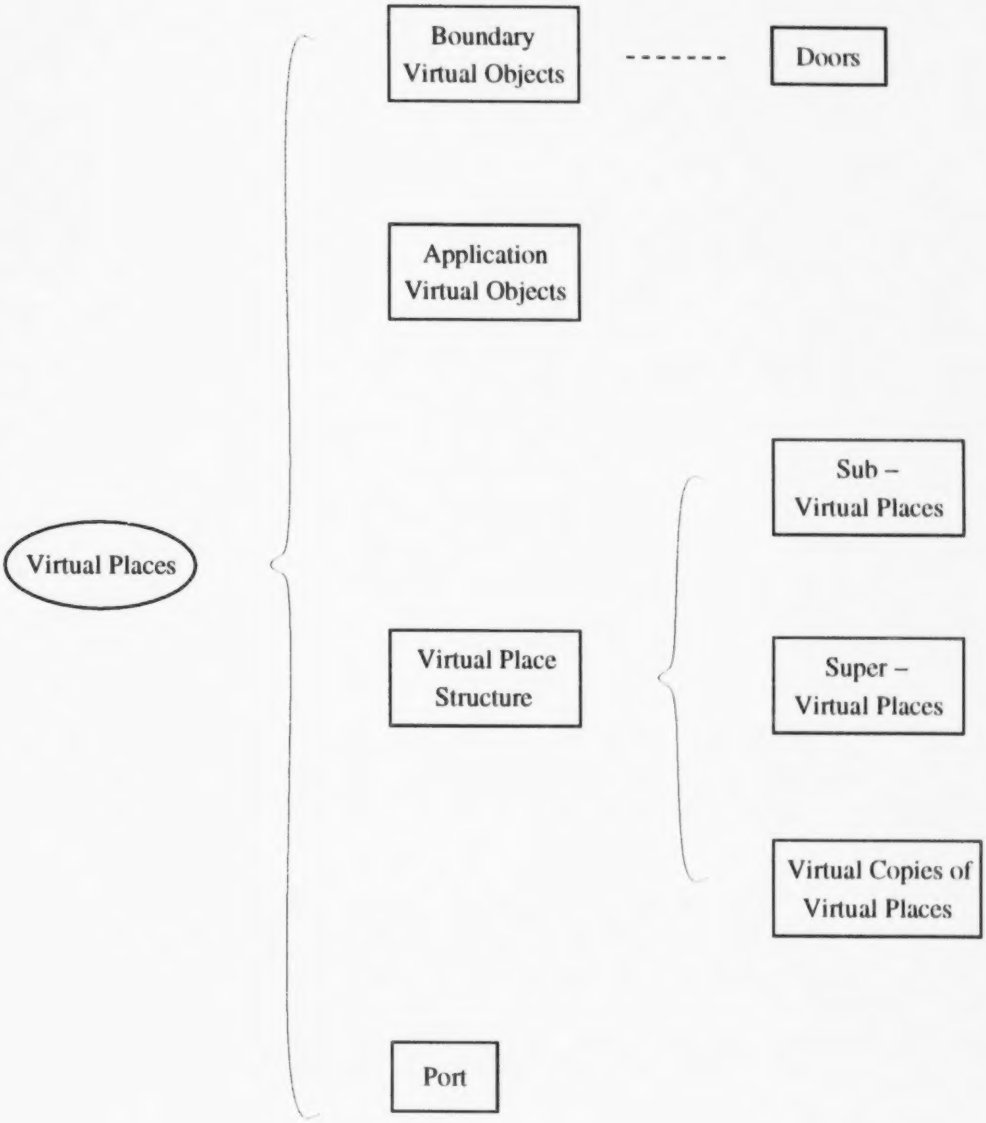

Figure 3.2: The Place Model

place.

A VP can contain any number of VOs, including virtual copies of VOs. Every VO in the virtual environment has a containing VP in the virtual world. This containing VP is called the current VP of the VO. The current VP of the table is the kitchen, and the current VP of the donuts is the oven. Since the user is a special type of VO, the user also has its current VP at any specific moment. The user's current VP is where the real user can interact with all the other VOs contained in the same VP, as well as the boundary of the current VP.

A VP can also contain many sub–VPs. A VP which contains sub–VPs is called a super–VP. The house is the super–VP of the kitchen and the bedroom. There is no living room as a VP. The living room is the space outside of the kitchen and the bedroom, and within the boundary of the house. This hierarchical structure of VPs can be very effective in organizing a medium–scale or large–scale VR environment. It helps to improve programmers' productivity by dividing a large, complicated application environment into different levels of VPs. It could also be very helpful for the user to get familiar with a new environment, since studies of spatial cognition reveal that people tend to remember one place within another. The path through the virtual world to the user's current VP is very useful in providing help information about the location of the user. Theoretically, there is no limit to how deeply one VP can be nested within another. However, too many levels in a VP structure would only be confusing rather than helpful.

To provide bidirectional sub–VP to super–VP navigation for the user, there can be a special sub–VO called a door somewhere in the boundary object hierarchy. There is a door as a sub–VO of each boundary of the kitchen, the bedroom and the house. The doors' event–handling behavior can decide if the doors should be open. Different kinds of doors could have different conditions to be opened. If the open condition is satisfied, the door should be gradually opened for the user. Then the user can navigate through the door to the VP on the other side of the door.

To provide bidirectional navigation between two neighbor VPs, which are not sub–

VP and super–VP, one of the VPs could have a door sub–VO as part of its boundary. The other VP can use a virtual copy of that door as a sub–VO of its boundary, i.e., the two VPs share the same door. This is one of the uses of a VOs' virtual copies. When the user is navigating through the door, he or she is going from the current VP to the other VP instead of going from the sub–VP to the super–VP or from the super–VP to the sub–VP.

Doors provide users with their most familiar way to navigate among VPs. An alternative approach, commonly used in other 3D graphics animation systems and VR systems, is to allow users to fly freely in the whole environment, through any VP boundaries. An application environment may provide both navigation facilities. But every boundary object of all the VPs in this environment must have an event–handling behavior which detects when an user crosses the boundary and sets the correct current VP for the user.

VPs may have multiple virtual copies of themselves, too. We call these virtual copies crystal balls because they provide views to their original VPs. Crystal balls make it possible for a user to observe what is happening in one or several other VPs at the same time while working in the current VP. In the example environment, a crystal ball of the oven can be put in the bedroom so that users will know whether the donuts are cooked or not from the bedroom. The difference between the virtual copy of a VO and the virtual copy of a VP is that a virtual copy of a VP will be restricted to viewing instead of interacting. The user will not be able to interact with any VOs within the original VP. The reason is that usually the virtual copy of a VO will have the same size of the original VO, while the virtual copy of a VP will be much smaller than the original VP in order to fit into the user's current VP as an ordinary VO. To select and manipulate objects in a small space is not an easy job for the user. If the user really wants to interact with VOs in that VP, the crystal ball provides a jump–in facility which allows the user to navigate from the current VP to the VP of the crystal ball. Once the user jumps in, there will be another crystal ball of the user's previous VP in his or her current VP. This crystal ball will let the user jump back to the previous VP, if he or she wishes to. Then the user can do the interactions in this current VP.

The jump–in facility provides an alternative navigational tool that saves an user's time in navigating back and forth between the current place and the original place of the crystal ball. Moreover, in situations where a door is unnatural, the user can either fly over the border to a neighbor place, or make use of crystal balls to jump into whichever place he or she wants to go. The programmer may also provide a 3D menu for the user to choose the place to go, then simply remove the user from the current place and put him or her in the new place.

Because a VP may have a number of VOs and sub–VPs in its space, when a user requests to enter the VP, it is possible that the user is put in the middle of a VO, or most likely, in a sub–VP of the VP. For example, if a user asks to enter the house in the example environment, the user may find himself or herseif in the kitchen or the bedroom, while he or she really wants to enter the living room. In order to avoid such situations, each VP must have a specified space which is not occupied by any VOs or sub–VPs at any time. This specified space is called the Port of the VP.

Chapter 4

Multi–User VR System Architecture

In this chapter, a multi–user VR system architecture is proposed to allow more than one user to share the same virtual environment, and these users do not have to enter the environment at the same time or quit at the same time. A two–user example application system is used to demonstrate the architecture. The two users, user A and user B, share the example house environment discussed in the previous chapter.

Since slow response time is a big problem for multi–user VR systems, the replicated architecture is used to achieve better response time as well as lower network traffic. Each user has a local copy of the application. The input from the local user is processed in the local application. Only the results are sent to other users to ensure that all users have the same view of the shared environment. The output to the local user is obtained from the local application. Therefore, the response time for the local user is not affected by network delay. The network traffic is also lower than when a centralized architecture is used.

The local databases of all users have to be synchronized because of the replicated architecture. When a user joins a shared environment, information about shared users must be available so that the new comer can establish connections with them, and get the most up–to–date data to update the local database. When the last user quits from the environment, the changes made by all shared users should be saved somewhere, so that the next user can continue from this point rather than start from scratch. Therefore, a session manager is needed to manage the users' activities and their database transactions. Each user also needs a communication component to communicate with the session manager and other shared users.

Figure 4.1 shows the architecture for multi–user VR systems. The figure is drawn for a two–user application. If there are more than two users, the architecture is the same except that the VR Session Manager deals with multiple users rather than two.

The VR Session Manager helps shared users establish communications and coordinates the database updating for late comers. Each user's application has a Communication Component in addition to the four components in the Decoupled Simulation Model. The Communication Component manages communications with the VR Session Manager as well as data synchronization with the shared users.

The Decoupled Simulation Model is used because the Computation Component can be a separate process which helps to reduce the response time for every user. The model is enhanced such that the Geometric Model Component uses the hierarchical Virtual Object Model and Virtual Place Model to represent the application objects and application environment. The Interaction Component gets input from the local user then sends the input to the Computation Component. The Computation Component does the object behavior evaluation according to the scheduling algorithm, sends results to the Geometric Model Component and the Communication Component. The Communication Component informs the Communication Components of its shared users to update their local databases. The Presentation Component selects all the objects which are possibly visible to the user at the moment, and presents them to the user.

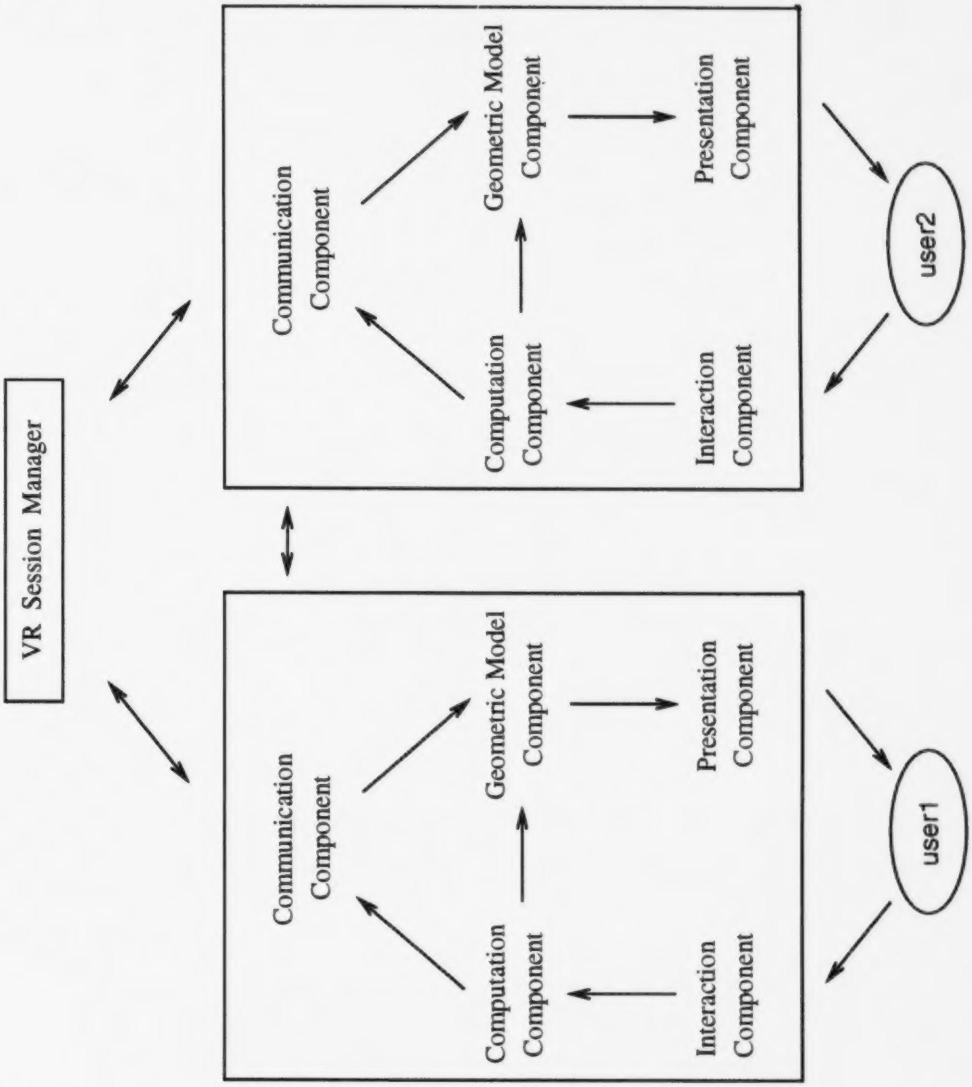

Figure 4.1: Multi–user VR system architecture

4.1 The VR Session Manager

The VR Session Manager is responsible for introducing a new user to the shared environment, i.e, it helps the new user establish communications with related shared users and coordinates the database updating for the new user.

A user list is maintained at the Session Manager's site. The list records each user's addresses, including the host name and port number, as well as the user's viewing places and viewing objects, which are visible to the user from the user's current place.

Whenever the Session Manager receives a request from a new user to enter the shared environment, the user is put into the user list. The Session Manager can compare the new user's viewing places and viewing objects with other user's viewing places and viewing objects and find the shared users for the new comer. It can then help the new user establish connections with these shared users and coordinate updating the database for the new user. In this way, only the shared users are connected to the new user and the new user can get all the updated data he or she needs from these shared users.

The VR Session Manager is also responsible for keeping the changes when there are no users in a changed place. A transaction list maintains all modifications to all places up to the time when the last user leaves each shared place. When the next user comes into a changed place, the corresponding transactions will be dispatched to update that user's database.

For example, when the first user, userA, enters the house, the Session Manager adds userA into its user list. Because the house has a window, userA's viewing places include not only the house but also the outside of the house — the virtual world. Suppose the second user, userB, enters the virtual world after userA dragged the chair to a new position in the house, since userB's viewing places are also the virtual world and the house, the Session Manager will inform userB to connect to userA so that they can synchronize their databases as more changes are made. UserA will also be instructed to send the chair's current position to userB.

If userA leaves the example environment after userB, the Session Manager will request userA to send over the chair's new position and save it in the transaction list. The next time either userA or userB goes into the house, the Session Manager will transfer the chair's position to that user.

4.2 The Communication Component

At each user's site, the Communication Component communicates with the Session Manager and performs data synchronization with the local user's shared users.

Similar to the Session Manager, the Communication Component also has a shared user list and a transaction list.

The shared user list records the addresses of the Session Manager and shared users for the local user to initiate communication with them. It also keeps viewing objects and viewing places of the shared users. The shared users for the local user refer to the users who share at least one virtual object or one virtual place with the local user, that is, every shared user has at least one viewing object or viewing place which is the same as the local user's. Whenever the local user modifies a virtual object, the Communication Component selects the users that can see the changes from the shared user list and sends modifications only to the selected users. In this way, the network traffic can be further reduced.

When the local user enters the shared environment, the Communication Component obtains the information about its shared users from the Session Manager, puts them into the shared user list, and establishes connections with them. As new users join in and some shared users leave the environment, the Communication Component adjusts the shared user list accordingly.

When the user switches from the current place to another place, the viewing objects and viewing places will all change. Consequently, the shared user lists have to be changed too. The Communication Component can simply send a request to the Session Manager to remove the user from the current place and re–register the user in the new place. A more complicated approach is to get the viewing objects and viewing places for the new place, send them to the Session Manger, and get the new list of shared users. The Communication Component can compare the new shared user list with the current list, disconnect with those users who are not in the new list and connect to the users who are in the new list but were not in the current list. The updated data for the new viewing objects and viewing places can then be requested from the new shared users. This approach is more efficient when some of the viewing objects and viewing places of the new place are the same as those of the current place.

The transaction list serves the same purpose as the one kept by the Session Manager. It saves all the most recent changes made to the local database in order to update databases for late comers. Whenever the Communication Component receives a request to update a particular object or place for a new–comer, it will send out the changes to that object or place from its transaction list.

The transaction list is expanded whenever the local environment is updated, either by the local user or by a shared user. When the local user modifies an object, the Communication Component not only sends the transaction to the shared users but also saves it in the local transaction list. If the object had been modified before, the previous transaction is replaced by the new transaction so as to control the growth of the transaction list. When a shared user sends updated data for an object, the Communication Component also saves the information in the transaction list and then delivers it to the Geometric Model Component to update that object.

Before the user quits from the current place, the Communication Component will check to see whether every transaction in its transaction list has another copy in a shared user's site. If not, the transaction will be transferred to the Session Manager and saved in its transaction list.

Take userA's Communication Component as an example. When userA enters the house as the first user in the environment, its shared user list has only one item: the Session Manager. The transaction list is empty. As userA drags the chair around, the chair's new position is recorded in the transaction list. Once userB joins userA in the virtual world, userB is added into userA's shared user list because userB's viewing places, the virtual world and the house, are the same as userA's viewing places. The Session Manager will also instruct userA to help update userB's database. UserA's Communication Component will get the chair's current position from its transaction list and send it to userB. If userA moves the chair again while userB is still in the environment, the Communication Component will save the chair's position in the transaction list and then send it to userB, who can see the changes. When userB quits from the virtual world, userA removes userB from the shared user list. Therefore, when userA exits from the house, the Communication Component knows the local user is the last user in the house, and sends the chair's most recent position to the Session Manager before terminating the program.

4.3 The Geometric Model Component

The Geometric Model Component maintains a high–level representation of the virtual world. According to the virtual environment model described in the previous chapter, the virtual world is a hierarchy of places. Each place may contain a number of application objects. Each application object can be an object composed of many sub–objects in its object hierarchy. The Geometric Model Component contains the structural information and physical properties of all the virtual objects, including application objects and place boundary objects as well as the local user object. The Presentation Component needs this information to display the environment to the end user.

The Geometric Model Components for userA and userB are identical, except the local user objects. Both components store the place and object hierarchical structure, such as the world contains the house, which contains the kitchen and the bedroom, the clock is composed of the frame, dial markers, and hour hand, minute hand and second hand. Part of the structural information is the positions and orientations of the places and objects relative to their containing place's coordinate system. The Geometric Model Components also store the shapes, sizes and colors of the objects as well as the walls of the places.

4.4 The Interaction Component

The Interaction Component collects input from the local user. The input, including eye and hand position and orientation, is transformed from the virtual world coordinate system to the coordinate system of the user's current place. They are then sent to the Computation Component to compute the feedback.

For example, if userA is in the house, userA's eye and hand position and orientation will be transformed from the virtual world coordinate system to the house coordinate system, so that userA and other objects in the house will be in the same coordinate system, thus userA can manipulate those objects including the chair.

4.5 The Computation Component

The Computation Component contains a collection of object behaviors. It obtains input from the Interaction Component, and evaluates the object behaviors according to the scheduling algorithm, which is the behavior culling algorithm as described below.

Since the user can only manipulate objects in the current place, only the event–handling behaviors of these objects are evaluated. The event–handling behaviors of the boundary objects of the current place as well as its sub–places are also invoked so that the user may navigate through doors to other places. For instance, userA is in the house, thus he or she can only interact with the clock, chair, sofa, go through the house door to the outside, go through the kitchen door to the kitchen, or go through the bedroom door to the bedroom. Therefore, the event–handling behaviors of the objects in the kitchen and the bedroom need not to be evaluated.

In case more than one user is trying to modify an object at the same time, depending on the application, either a concurrency control mechanism can be used to allow only one user to modify the object or the programmer can get a compromised result by considering all users' modifications. Concurrency control is an important problem, but it lies outside the range of this thesis, so it won't be considered here.

The Computation Component also determines which animation behaviors of which objects in the virtual world should be evaluated. In order to achieve better response time, we assume that the ordinary animation behavior of any object which is not visible to the user can be suspended. The Computation Component will only evaluate behaviors of those objects which are possibly visible to the user at the present time. Suppose userB is in the kitchen, the Computation Component of userB will not evaluate the clock's animation behavior because userB can not see the clock anyway. The always–active behaviors of any objects in the virtual world are evaluated in every frame. The donuts are always being baked in the oven until they are cooked, even if no users are present.

The results of the behavior evaluations are sent to the Geometric Model Component to update the local database and to the Communication Component to update shared users' databases.

The behavior culling algorithm saves the event–handling behavior evaluation time and ordinary animation behavior evaluation time of those objects which are not visible to the local user, which reduces the overall response time of the VR environment.

4.6 The Presentation Component

The Presentation Component gets input from the Geometric Model Component, produces the visual, sonic or force display, and presents them to the local user.

The Presentation Component is composed of display procedures for all the virtual objects. Each display procedure is responsible of converting the data representation of the physical properties to the visual, sonic, or force display for its object. Similar to the Computation Component, the Presentation Component contributes to the improvement of response time by presenting only those objects that are possibly visible to the user at the present time. For instance, userB's Presentation Component only presents objects in the kitchen plus the kitchen's boundary, while userA's Presentation Component presents everything but objects in the kitchen and the bedroom.

Chapter 5

Multi–User Place System

The Multi–User Place System supports the Virtual Object Model, the Virtual Place Model, and the Multi–User VR System Architecture. It provides a framework for application systems to construct hierarchically structured environments with hierarchically structured application objects in them. Facilities such as link object, crystal ball, and door are also provided. Communication mechanisms including the Session Manager and the Communication Agent are embedded in the Multi–User Place System to assist communications among shared users. An analogy between the traditional user interface software system and the proposed VR system model is made, showing how existing software techniques can be extended to VR.

5.1 System Overview

As shown in Figure 5.1, there are five layers in the Multi–User Place System. The upper layers can make use of the facilities provided by any of the lower layers.

The bottom layer contains Object, which supports hierarchically structured objects in a virtual environment, including object transformation, behavior evaluation, and object presentation.

The second layer contains Link Object, Place, and Door. The Link Object provides an exact copy of an object in a different location, enabling the user to manipulate

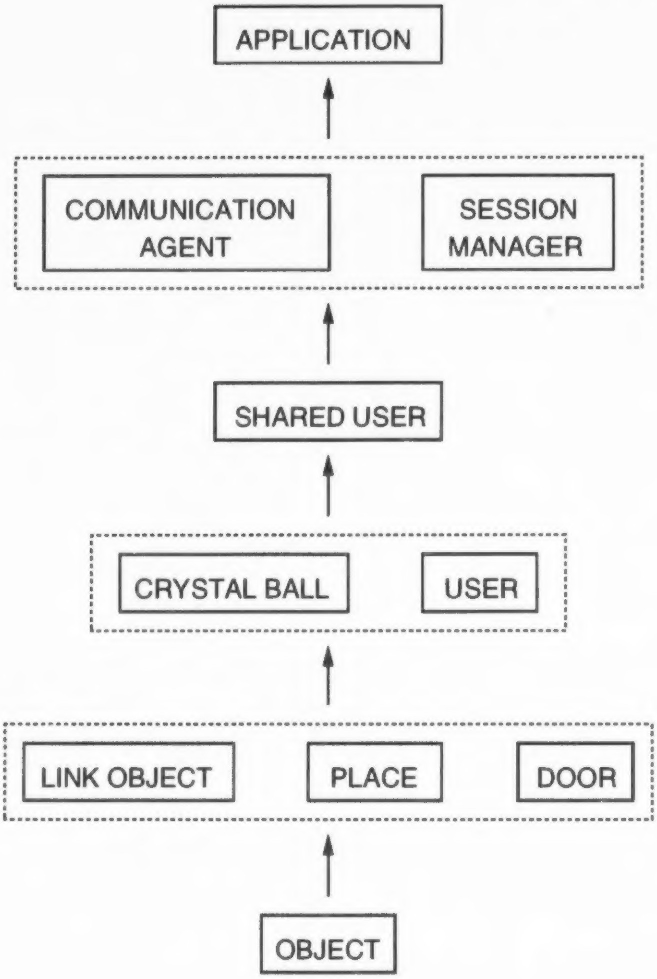

Figure 5.1: Multi–User Place System

an object that is not in his or her current place. It can also be used to provide the user with multiple views of one object. Place organizes a virtual environment using an hierarchical structure. Response–time improvement and network traffic reduction are based on this hierarchical environment structure. Door is a special type of Object that provides a navigation tool for the user to go from one place to another.

The third layer includes Crystal Ball and User. The Crystal Ball provides a small version of a Place so that the user can observe what is happening in that place. The Link Object is used in its implementation. The User gets the local user's eye and hand positions and orientation, and converts them into his or her current place's coordinate system. It also does behavior and display culling according to the current place of the local user as well as information about the virtual world.

In the forth layer, there is Shared User. Shared User keeps the communication related information for a User. It provides facilities for the local user to send or receive messages to or from that shared user. It also keeps the viewing objects and viewing places of the shared user, thus the Communication Agent is able to decide if that user can see a local change made to a particular object and whether to send the updated data to that user.

The fifth layer is composed of Communication Agent and Session Manager. The Communication Agent communicates with the Session Manager and the local user's shared users. It is responsible for registering the local user with the Session Manager, receiving its Shared Users' information from the Session Manager, and synchronizing the local database with its Shared Users. When the local user is leaving the virtual world, the Communication Agent also informs the Session Manager and all its Shared Users. The Session Manager coordinates Users' entering and leaving the virtual environments, helps them to establish communications and update databases for each other.

The top layer is VR applications. The VR applications can use or extend any facilities provided by the Multi–User Place System in the construction of its virtual environment.

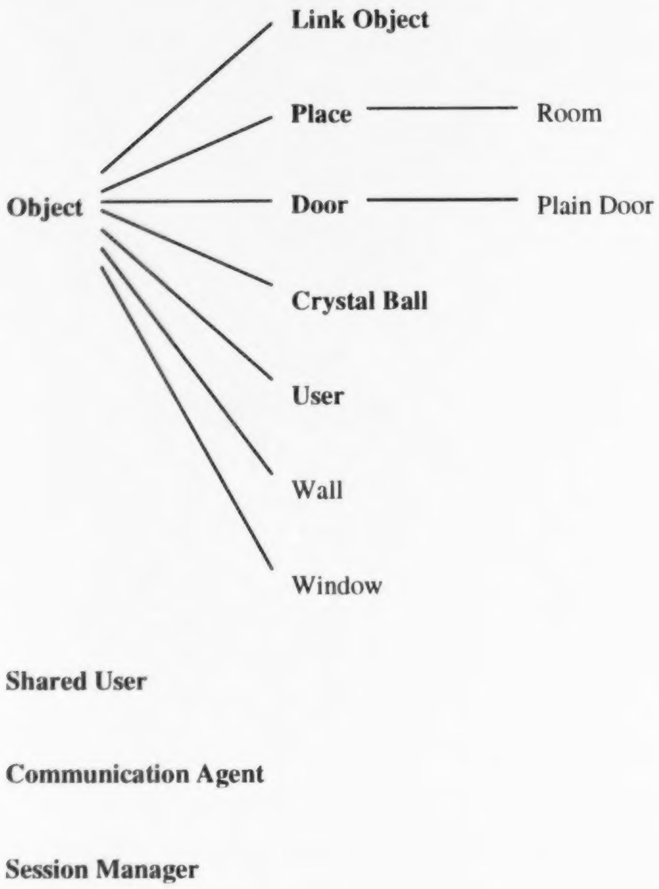

Figure 5.2: Multi–User Place System Class Hierarchy

5.2 Class Hierarchy

The Multi–User Place System is implemented in the object–oriented language C++.

Figure 5.2 shows the class hierarchy of the Multi–User Place System. VR programmers can use or extend any class to build their own virtual worlds. The classes in plain text are examples of extended classes.

5.3 Object Class

The Virtual Object Model presented in chapter three is used in the implementation of the Object class.

5.3.1 Physical Properties

An object may have many different object–dependent physical properties. But there are some common properties that every object must have. These common properties are:

1. The size of the object's bounding box
2. The object's location
3. The object's material and color

Bounding Box

The bounding box of an object in the Object class is defined by the origin of the box and its length, width, and height. The Object class provides the following functions for the bounding boxes:

- Set or get the origin and sizes in x, y and z directions in the object's coordinate system
- Test if a single point in the object's coordinate system is within the bounding box
- Decide the area a point is in relative to the bounding box (left, right, front, back, bottom, top or in)
- Calculate the intersection point on the bounding box of a line segment from a point's previous position to it's current position

The bounding boxes are useful in improving the efficiency of collision detection. For example, when userA is in the house, he or she should not be able to go through any wall of the house, the kitchen or the bedroom. To detect whether userA is trying to go through a wall to the bedroom, userA's current eye position is transformed to the bedroom's coordinate system, and then tested if userA is in the bounding box of the bedroom. If not, there is no need to test userA against every wall of the bedroom. For the house, since userA is in the house, the eye's current as well as previous position are checked against the bounding box of each wall of the house to see if both positions are on the same side of the wall. If the result is no, the whole virtual world must be translated in the direction of userA's movement so as to prevent userA from going outside of the house. The distance the virtual world must be translated should be a little bit greater than the distance between the intersection point where the user is supposed to hit the wall and userA's current eye position.

Depending on the object, its bounding box could be any irregular shape. VR programmers can extend the Object class and overload the above functions according to the shape of the new bounding box.

Location

By location we mean the place the object is in, and the object's position and orientation in that place's coordinate system (if the object has no super–object), or the object's position and orientation in its super–object's coordinate system. The position and orientation of an object can be represented by a 4x4 transformation matrix.

The following functions are available for specifying and changing the object location:

- Set place for the object
- Get the current place the object is in
- Reset the object's matrix to the unit matrix

- Set the object's matrix to a given matrix
- Get the current transformation matrix
- Get the inverse of the transformation matrix
- Scale, rotate, or translate the object
- Transform the object using a given matrix

Material and Color

Programmers can set material, set RGB color, get the current RGB color, or check if the material and the color are set for the object. The default is not set. In this case, the material and the color of the object are assumed to be the same as those of its super–object. Therefore, for an object which has most of its sub–objects in the same color, the programmer does not have to assign the color to each of these sub–objects. The programmer can simply assign the color to the object and set the colors for other sub–objects individually. For example, all the sub–objects of the chair are in a wooden color except its back. Thus the chair object can be set to be in wooden color while the back of the chair is set to be in a different color.

5.3.2 Structure

An object can be composed of one or more sub–objects. Because of the hierarchical structure of the object, conversions among different coordinate systems are required. Suppose userA is in the bounding box of the bedroom when we detect that he or she is trying to go through the bedroom walls. To do this, userA's coordinates have to be converted from the house's coordinate system to the bedroom's coordinate system, then to the coordinate system of each wall of the bedroom to test which wall the user is trying to pass through. Therefore, a point in a sub–object's coordinate system may need to be converted to the coordinate system of its Top Object (the object at the root of the object hierarchy, which has no super–object), or converted from the top object's coordinate system to a sub–object's coordinate system. In a different situation, the point may have to be converted between a sub–object's coordinate system and its place's coordinate system. By making use of these conversion functions, we can also test if a given point in a top object's coordinate system is within a sub–object's bounding box.

The Object class provides the following functions to support the hierarchical object structure:

- Add a sub–object to this object
- Remove a sub–object from this object
- Convert a vector or a point from the coordinate system of this object to the coordinate system of its top object
- Convert a vector or a point from the coordinate system of this object's top object to its coordinate system
- Convert a vector or a point from the coordinate system of this object to the coordinate system of this object's current place
- Convert a vector or a point from the coordinate system of this object's current place to the coordinate system of this object
- Get the transformation matrix for any of the above conversions
- Test if a point in a given top object's coordinate system is within the bounding box of this object

An object can also be linked by a Link Object, producing another copy of the object at a different location. This results in multiple super–objects and multiple top objects for this object and its sub–objects. For instance, when there is a door between two neighbor places, say room1 and room2, the door in room2's boundary object hierarchy is a link object of the door in room1's boundary object hierarchy.

Consequently, the door has two top objects: one is room1's boundary object, the other is room2's boundary object. Depending on which room the user is in, the user's coordinates will be transformed from that room's boundary object's coordinate system to the door's coordinate system to see if the user is close enough to the door then decide if the door should be open.

The following functions are also provided by the Object class:

- Add a link object to this object
- Remove a link object from this object
- Get the top object of this object
- Get the next top object (connected by a link object) of this object

5.3.3 Behavior Evaluation and Presentation

The Object class provides three functions to handle three different types of object behavior: event handling behavior, animation behavior, and always active behavior. These functions traverse the object hierarchy from the current object down to the bottom level of the hierarchy, i.e. they simply call their sub–object's functions.

For any application object, typically at the bottom of the object hierarchy, the programmer extends the Object class and overloads these functions with the special behaviors of the application object. In cases where an object has its own behavior and it also has sub–objects, the programmer must extend the Object class to evaluate its own behavior as well as it's sub–objects' behaviors. For example, the clock's animation behavior is overloaded to obtain the time from its host computer and invoke the animation behaviors of the hand pointer, minute pointer and second pointer.

The presentation of the object presents all of its sub–objects. Each sub–object is transformed first, by using the sub–object's transformation matrix. If the material or the color is set for the sub–object, the current material and color are then saved and this sub–object's material and color are used for drawing from this point. The saved material and color are recovered after this sub–object's drawing is done. Otherwise the current material and color will be used for drawing the current sub–object. In the case of the chair, before the back is drawn, the wooden color for the chair object is saved and the color for the back is used to display the back. Then the wooden color is recovered to draw the remaining sub–objects of the chair.

For an application object, the presentation function must be overloaded by its drawing procedure.

5.4 Link Object Class

The Link Object class is a sub–class of the Object class. Therefore, a link object can be put in any position in an object hierarchy.

We have to consider the following two problems during the implementation of the Link Object class:

1. The transformation matrix of the original object must be replaced by the transformation matrix of the link object whenever we traverse the link object's hierarchy

2. A link object could cause repeated evaluation of its ordinary animation behaviors

5.4.1 Transformation Matrix

The functions manipulating the transformation matrix provided by the Link Object class take the transformation matrix of a link object and ignore the original object's transformation matrix. Therefore, we can treat a link object exactly like an ordinary object whenever a link object is encountered while traversing an object hierarchy. For example, when the clock link object in the bedroom is displayed, the transformation matrix of the clock link object, specifying its position and orientation in the bedroom's coordinate system, is used; while the transformation matrix of the original clock that specifies the clock's position and orientation in the house is ignored.

5.4.2 Animation Behavior Evaluation

The evaluation of a link object's event handling behavior and the presentation of the link object are simple. We just invoke the corresponding functions of the original object.

For ordinary animation behavior of a link object, we have to consider the possibility of repeated evaluation. For instance, we may have two copies of an object (one original object, one link object) showing a simulation process from different angles to the user. In this case, repeated evaluation may not be acceptable. If the object is supposed to move one step per frame, repeated evaluation will result in two step movements of the object per frame. In order to prevent this from happening, the Object class's regular animation evaluation function is overloaded. The Link Object's regular animation evaluation function tests if the original object has been evaluated before doing the evaluation.

The always active behavior of the original object should be evaluated in its object hierarchy. Therefore the link object's always active behavior evaluation is overloaded by an empty function.

5.5 Place Class

The Place class is a sub–class of the Object class. The Virtual Place Model is used to implement the Place class.

5.5.1 Physical Properties and Structure

A place itself is an object. Its boundary objects are its sub–objects. The appearance and behaviors of a place are decided by its boundary objects. A collision avoiding function is provided for preventing objects from going through the boundary objects.

Doors could be sub-objects in the place object hierarchy so that the user can go in and out of the place by a door.

A place may contain objects and sub-places. VR programmers can use the following functions to put application objects in places and build their own place hierarchy:

- Add an object to the place
- Remove an object from the place
- Add a sub-place to the place
- Remove a sub-place from the place

Now we have the place hierarchy, we can provide functions for converting between the coordinate system of this place and the coordinate system of the whole virtual environment.

- Convert a vector or a point from this place's coordinate system to the environment's
- Convert a vector or a point from the environment's coordinate system to this place's coordinate system
- Get the transformation matrix for the above conversions

The first function is used when userA is trying to go through a wall of the house. The virtual world has to move away to prevent it. But the vector from the point where userA is supposed to hit the wall to userA's current position is in the coordinate system of the house, not the virtual world. This vector has to be converted to the coordinate system of the virtual world before it is used to translate the world. The second function is used to convert the local user's eye and hands position and orientation from the environment's coordinate system to the user's current place's coordinate system.

One special property of places is whether they are open. If a place is open, the user will be able to see outside the place from inside, or see inside the place from outside. In the case of closed places, the user can only see the objects that are inside of the place. We use this property to decide the visibility of places to the user. If the current place is open, it is possible that the user can see objects in all the open super–places along the current place hierarchy until we find a closed super–place. This closed super–place is called the Top of the open places. The presentation of the virtual world starts from the Top of the open places of the local user's current place, and displays all the open sub–places and objects in them along the place hierarchy. Several functions are provided concerning the open property:

- Open the place
- Close the place
- Check to see if the place is open or closed
- Get the Top of the open places of the current place

Another property of places is port. The position and size of a port defines a space which is guaranteed to be free in the place. That is, the space of a port must not be occupied by any objects or any sub–places in this place at any time. Ports are used to avoid confusing situations such as transporting the user to a sub–place while the user is meant to be transported to the super–place of that sub–place. The port of the house must not be in the kitchen or the bedroom, or in the same position as the chair or sofa. Two functions are provided for setting and getting a port.

- Set the position and size for a port
- Get the position and size for a port

The default port position is at the center of the place.

5.5.2 Behavior Evaluation and Presentation

The user should be able to interact with the boundary objects of the current place, the boundary objects of the sub–places and all objects located in the current place. The evaluation of the event–handling behavior of a place invokes evaluations of event handling behaviors of all the above objects. For example, the event–handling behavior of the house invokes the event–handling behaviors of the boundary objects of the kitchen, the bedroom, as well as the house, allowing the user to go through any door to another place, and to stop the user if the user tries to go through any of the walls. The event–handling behavior of the clock, chair and sofa are also called, thus the user can move any of them around.

The animation behaviors of all objects that are possibly visible from the user's current place, including the objects and boundary objects of the user's current place, plus its sub–places, should be evaluated. If any of the sub–places is an open place, its objects and sub–places should be evaluated using the same strategy. The objects in a place with an open door to the current place should also be evaluated. However, infinite recursions could occur in cases where the current place has an open door to the next place, which has an open door to the third place, which has an open door to the current place. Such cases have to be detected and evaluation terminated once it happens. Just as we said in the object section, we do not want to repeat animation evaluation of any object in one frame. It could result in undesired effects.

Similar to the animation behavior evaluation, all possibly visible objects have to be presented to the user. In order to prevent infinite recursions in each frame, the place is tested before being rendered and marked after the rendering is done.

5.6 Door Class

The Door class is also a sub–class of the Object class. It provides a navigation facility for the user to move from one place to another while exposing the places on the other side of the door to the user.

5.6.1 Navigation Related Functions

The following functions determine the policy as to when and how the door should be opened or closed, and when and how the user should be transported from the current place to the other side of the door.

- Test if the door should be open
- Test if the door should be closed
- Animate the door opening process
- Animate the door closing process
- Initialize user transportation
- Transport the user from the current place to the other side of the door

The functions that help to implement the Door's behaviors are:

- Set the door's status to opening, open, closing, or closed
- Check the door's status
- Test if the user is inside a place or outside
- Find out which place is on the other side of the door

The default policy offered by the Door class is:

- Whenever the user is close enough to the door, the door should be open. Otherwise it should be closed.
- The door will gradually open towards outside or inside (by rotating along its axis, which is its left or right edge) depending on which direction the user is going. The door will be open towards outside if the user is going from inside the door to the outside of the door, and vice versa. Otherwise, the user would back away from the door to avoid a collision with it. If the user moves away from the door, the door will start to close.

- When the door is fully opened, the user is transported from his or her current position in the current place to the center of the door position, then to the place on the other side of the door. After the user reaches a position which is farther from the door, the door starts to close and the user will continue to be transported along the direction the user is looking until he or she decides to stop by actually walking physically in the real world.

The evaluation of the door's event–handling behavior invokes the testing functions to check whether the door should be open or not, and set the status for the door to be opening, open, closing, or closed. The animation behavior simulates the door opening, door closing, or user transport processes according to the door's status.

Different doors may use different policies. The programmer can extend this Door class, overloading any of the above functions to adopt a new policy.

5.6.2 Visible Places' Behavior Evaluation and Presentation

When the door is open, the user will be able to see what is happening in the neighbor place. When this place is an open place, its super–place and all open sub–places of the super–place are probably visible to the user too. So are all the super–places up along the place hierarchy until we find a closed super–place, which we call Top of the Open Place. All these visible places have to be rendered and their animation behaviors have to be evaluated.

The animation evaluation of the door evaluates the door animation behavior, as well as the animation behaviors of the Top of the Open places and those of its ancestors.

The presentation of the door also renders the Top of the Open Places besides the door.

5.7 Crystal Ball Class

The Crystal Ball class is a sub–class of the Object class. A crystal ball contains a small version of a place, allowing the user to observe what is happening in that place from the user's current place. We make use of the Link Object class to implement the Crystal Ball class. A link is made to the original place to produce the small version of it within the crystal ball.

Since the user only has a small version of the original place and due to the fact that device noise level is quite high, we do not allow the user to manipulate any objects from the current place. If the user needs to interact with an object in the original place, the Crystal Ball class provides a jump–in facility. The event–handling behavior of a crystal ball will transport the user from the current place directly to the port position of that place, if the user is within a certain distance to the crystal ball, looking at it, and the user's hand is pointing at it at the same time. Of course any programmer can extend the Crystal Ball class and overload the default.

The evaluation of the animation behavior of a crystal ball triggers the evaluation of the original place's animation behavior, since the user is able to see everything inside the original place.

The presentation of a crystal ball is a little bit more complicated than simply drawing a transparent sphere and the original place. In order to let the user see inside, we draw the place with backfacing place boundary objects removed.

One possible problem is that sometimes the original place is not drawn inside the crystal ball since it has been drawn before. For instance, a user is in a place with a crystal ball of the neighbouring place. When the door is open, the neighbouring place has to be drawn. But the mechanism that prevents infinite recursions will prevent this place from being drawn again inside the crystal ball. In order to solve this problem, we have to save the status of the original place and reset its status to be undrawn, then draw the original place, and recover the saved status for the place after the drawing is done.

5.8 User Class

Users are a special type of object. A user may have more than one sub–object depending on the VR devices locally available. The default user has two sub–objects: user's eye and a user's hand. Their position and orientation are controlled by the real user in the real world. Whenever new data comes in, they are converted to the coordinate system of the current place of the user from the environment's coordinate system. The Eye Object also provides functions to test if the user is walking or turning in the physical world according to the data reported by the device. The Hand drawing procedure provided by the MR Toolkit draws the hand in the environment coordinate system. The Hand Object provides a new function to draw the user's hand in the current place.

One big difference between a user and an ordinary object is that when the user is transformed (transported), he or she could be standing still in the real world. In order to make the user feel that he or she is being transported, the whole virtual world is transformed in the opposite way. The Scale, Rotate, Translate and Transform functions are overloaded in the User class. The transformation matrix in the user's current place is converted to the coordinate system in the virtual environment, and the whole environment is transformed by the inverse of the converted transformation matrix.

The event handling behavior of the User class tests if the user should start walking, stop walking, start flying, or stop flying. By walking we mean the user is transported on the x-y plane along the direction the user is looking. By flying we mean the user is transported in the direction the user is looking. A function for setting the transport speed is available.

If the user is in the walking or flying state, the animation behavior evaluation obtains the user's eye orientation and transports the user in that direction at the given speed.

When a virtual environment is built by using or extending the classes provided, the User class provides an EnteringWorld function to start the environment and keep it running. It also provides the LeavingWorld function to stop the environment whenever the user wishes to leave.

The EnteringWorld function initializes the environment first. It adds the user object into a specific place, which is the user's current place. The user along with all the related information is then registered with the Session Manager through the local Communication Agent. The local database will be updated and connections with shared users will be established by the Communication Agent when the registration is done. Now the real user is supposed to be within the sensing limit of all VR devices in the real world. When the environment begins to run, the user will be put at the port position of his or her current place.

The environment execution process is a loop. For each frame, requests from the Session Manager or shared users are handled first. Then we check to see if the user wants to exit from the environment. If the user does, the Leaving World function is called. Before any behavior evaluation, all input data is obtained from the local VR devices and processed.

Because the user can interact only with objects in the current place, the event-handler behavior of the current place is invoked. For ordinary animation behaviors, all possibly visible objects must be evaluated. The top of the open places of the current place is found and the ordinary behaviors are evaluated from the top place down the place hierarchy. The always active behaviors are evaluated from the root of the place hierarchy (the world) every frame. In this way we implement behavior culling and improve the response time of the environment.

Similar to the ordinary animation behavior evaluation, the presentation of the environment starts from the top of the open places of the user's current place.

The LeavingWorld function calls the Communication Agent to inform the shared users and Session Manager that the local user is leaving, then it terminates the program.

5.9 Shared User Class

The Shared User class keeps information about a shared user and offers the following facilities to communicate with the shared user:

- Set the shared user's ID, including user name, host name, and port number
- Get the username, hostname, or port number
- Establish connection with the shared user
- Close the connection with the shared user
- Send a message to the shared user
- Receive a reply from that user
- Receive a message from the shared user
- Send a message to that user

The Internet Transmission Control Protocol (TCP) is used in the communication among shared users. In order to simplify the message receiving procedures, the message sending procedures write the message size before the actual message and the message receiving procedures read the size first then read this specified length of message.

The Shared User class also keeps a record of a shared user's viewing places and viewing objects. Viewing places are all places which are possibly visible to the user from the user's current place, including the current place. Crystal balls, open doors, and open places are the reasons that a user may have more than one viewing place. Viewing objects are objects which have multiple copies in a viewing place. For example, if userB is in the bedroom, userB's viewing places are the bedroom and the oven because there is a crystal ball of the oven in the bedroom. UserB's viewing object is the clock which is a link object of the clock in the house. When userA is in the house, userA's viewing places are the house and the virtual world since the house is an open place and userA's viewing object is the clock. The viewing place and viewing object information are used to decide which shared users can see the local changes made to a particular object so that the updated data can be sent to them. These shared users are called coworkers. UserA and userB are coworkers because both of them can manipulate the clock.

The following functions are available for the Communication Agents and Session Manager to maintain their user lists:

- Set viewing places of the shared user
- Set viewing objects in each of the viewing places
- Set the shared user's information, including user ID and the above information
- Get the shared user's information
- Decide if a shared user is a coworker of this user
- Add a shared user as a coworker of this user
- Remove a coworker

5.10 Communication Agent Class

Communication Agents update the databases for shared users so that the environment looks the same for every shared user. At the same time, the network traffic should be kept as low as possible. In order to meet the above two requirements, the local changes should be sent only to the corresponding coworkers. Communication agents manage shared users and provide facilities for VR programmers to update shared data and define their own protocols among shared users. The transaction lists keep records of all the changes to the local environments. Requests from the Session Manager and other shared users concerning the shared user list and database updates are also processed by the Communication Agents.

5.10.1 Shared User Management

Three functions are available to VR programmers to announce the local user's joining or leaving the shared environment, or switching from one place to another.

- Register the local user
- Remove the local user
- Move the local user from the current place to another place

Registering the local user involves three steps. First, the communication agent collects the local user's information including user name, host name, port number, viewing places and viewing objects. The information is sent to the Session Manager and the Communication Agent waits for the reply. The Session Manager sends the information about the coworkers, so that the communication agent can set up the shared user list for the local user. The third step is to update the local database by executing the transactions received from the coworkers, or from the Session Manager if no other users have such information.

Removing the local user from the shared environment also involves three steps. The first step is to inform the local user's shared users to remove the local user as a coworker. The local Communication Agent will wait for confirmation to avoid any problem that could be caused by network delay. Otherwise, a shared user's message sent to the leaving user before the quit message is received might arrive after the leaving user's program is terminated. The second step is to inform the Session Manager that the local user is leaving the shared environment. If the local user does not have coworkers with the changes to any of the viewing places or viewing objects, the corresponding transactions are transferred to the Session Manager. If the local user is the last user in the shared environment, all changes made by always active behaviors are transferred to the Session Manager too.

Moving the local user from the current place to a new place is equivalent to removing the user from the current place, and registering the user in the new current place.

A two-user example illustrating the maintenance of the user list was given in the previous chapter.

5.10.2 Shared Data Management

Object behaviors change the objects in an application environment. Some of these changes, like those produced by event-handling behaviors, are made at only one user's site. This type of change must be broadcast to all coworkers immediately. The second type of changes, performed by ordinary animation behaviors, occur at all coworkers' sites, while changes made by always active animation behaviors occur at all shared users' sites. As long as these animation behaviors are initialized correctly, the coworkers should be able to see almost the same results, even if the changes are not sent out every frame. However, the changes must be saved in order to initialize a new user's behaviors. One potential problem is that the results of these animation behavior evaluations could be very different after a period of time due to different update rates of different users, or the time differences among different computers, even though the behaviors are initialized correctly. For example, if userA is in the kitchen watching the donuts cooking while userB in the bedroom doing the same thing by looking at the crystal ball, suppose the donuts should be cooked in 30 minutes, due to the time difference between the two computers, userA may see that the donuts are cooked after 29 minutes and userB may see them cooked after 31 minutes. In this case, the programmer should periodically force data synchronization for these behaviors.

As a result of the above three types of changes, we have three types of shared data. One is Regular type of data. They will be sent to all coworkers whenever the Communication Agent receives such a request. The chair's transformation matrix is a Regular type of data. The second type is called Animation type of data and the last type is AlwaysActive data. The donuts' colors are AlwaysActive type of data. The last two types of data will be saved, but not sent to any other coworkers when updated unless the programmer explicitly asks for it. The Communication Agent will send the data to update a new comer's database when it is necessary.

Before a programmer updates a shared data for the local user's coworkers, the shared data must be registered with all Communication Agents to get its internal name — its data ID number. The reason is that the same data in different users' programs is assigned different addresses in different processes. For a Communication Agent that receives an updated data to find the data's local address and store the data into that address, we have to associate the data's different addresses with an identical ID number so that the Communication Agent can find the data address in any user's process using the data ID number.

Two functions are available for data sharing:

- Register data (regular, animation or alwaysActive type)
- Update a registered Data

The data registering function keeps a record of the data type, the data address and the length of the data. It also records the DataOwner, which is the object the data represents. The DataOwner of the chair's transformation matrix is the chair. DataOwner will be used to decide whether the shared data is a part of any viewing objects, or whether the DataOwner is in any of the viewing places. The shared data can then be sent to the corresponding coworkers by the data update function.

The data registering function returns the next available data ID. Shared data that are registered before the application environment starts to run should have the same IDs at all user's sites as long as the order of their registration is the same. However, the order of data registration can not be guaranteed to be the same at different user's sites after the application environment starts running. If two objects, obj1 and obj2, are dynamically created by two shared users, userA and userB, at the same time, the shared data of obj1 and obj2 will obtain the same data ID from userA and userB's local Communication Agents respectively. When obj1 is created on userB's machine, and obj2 is created on userA's machine, they will get the next data ID. In this case, the order of data registration on userA's host is obj1's shared data followed by obj2's shared data; while the order of data registration on userB's host is obj2's shared data followed by obj1's shared data.

In order to prevent data ID conflict at environment execution time, one possible solution is to have a centralized mechanism residing in the Session Manager to distribute the next available data ID. However, the Session Manager will become a potential bottleneck if all the users are creating new objects and requesting data ID numbers for the objects' shared data. The response time at users' sites will be increased even if all the users are working in different places and do not realize what is going on.

Another approach is to distinguish shared data which are originally registered from different users' sites. Each user's new shared data have their own naming space. The whole naming space is divided into a number of blocks. The first block is designated for pre–defined shared data, which are defined before the application environment starts to run. Each of the remaining blocks is allocated for data created by one of the shared users, including the local user. If a shared data is dynamically registered at a user's site, it will get the next available data ID in the block allocated for that user from the local Communication Agent. The same data ID will be returned after the corresponding registration statements are called on the coworkers' sites. For example, both userA's and userB's Communication Agents have three blocks in their shared data name space: the first block is for pre–defined shared data such as the donuts' current colors, the second block is for userA's dynamically created object data, and the third block is for userB's dynamically created object data. Suppose userA and userB are both in the house and userA created a chair at the same time as userB created a sofa, the chair's shared data will get the first ID in the second block of the name space while the sofa's shared data will get the first ID in the third block of the name space on both users' computers.

The number of blocks in the name space depends on the maximum number of coworkers of the local user. In the case of only a few users share an application environment, they might all be able to see each other's work, and thus the number of blocks should be the number of users plus one (for pre–defined shared data). If there are thousands of users sharing an environment, the number of blocks should be the maximum number of coworkers allowed for a user plus two (one block for pre–defined shared data; one block for locally created shared data).

The data update function checks whether the given data ID is valid. If it is, the transaction is saved, so it can be used to update late–comers' databases. The shared data is then sent to all coworkers of the dataOwner, if it is a regular type of shared data. If a programmer explicitly requests animation data or alwaysActive type of data to be updated for other users, the Communication Agent will send the data to the corresponding coworkers.

5.10.3 New Protocols

VR programmers can use the functions below to define new protocols for shared users.

- Define a new command
- Invoke the new command

The command defining function records a programmer–specified procedure and the place where the procedure will produce changes. The place is used to find corresponding coworkers for the changes. Similar to shared data in different processes, the specified procedure also has different addresses in different processes. A unique command ID number is returned for the programmer to refer to the command later.

The function invoking the new command will call the specified procedure locally as well as at the coworkers' sites.

VR programmers can use this feature to create or remove an object dynamically. If the created object is shared by several users, its shared data must be registered at all the users' sites. An easy way to do this is to put the shared data registration statement inside the object creation procedure. Or, the programmer can define another new command just for shared data registration and call the command after the object is created. The data registering function will return the same data ID to all the shared users.

5.10.4 Transaction Management

Transaction management is transparent to VR programmers. Whenever a shared data is updated, or a new command is called (such as shared object creation deletion), the transaction is saved at the local user's site as well as the coworkers' sites. The saved transactions will be used to update databases for new comers.

Typically, shared data is updated continuously for a number of frames. For instance, when a user is dragging an object to a new position, the matrix for the object is updated for all coworkers until the user releases the object. The Communication Agent will only store the most recent version of the shared data so as to save memory space.

When a new comer joins in the shared environment, if the new user is a coworker of the local user's viewing places or viewing objects, the local Communication Agent will be asked to transfer the corresponding transactions to that new user. Indexes are built for each of the viewing places and viewing objects so that the Communication Agent can find their transactions.

5.10.5 Request Processing

Every Communication Agent checks for requests from the Session Manager or from Communication Agents of the shared users, and processes these requests accordingly. The following requests can be processed by a Communication Agent.

- Add a shared user
- Remove a shared user
- Update shared data
- Call a new command

When a new user sends a register request to the Session Manager, if the local user is a coworker of the new user, the local Communication Agent will get a request from the Session Manager to add the new shared user into the local user list and send the required transactions to the new user.

A request to remove a shared user will be received if the shared user is leaving the environment. The Communication Agent will remove the user from the user list and send a confirm message back to the leaving user.

When a data update request is received from a shared user, the Communication Agent will find the local data address according to the given data ID number and place the new data in that address. The transaction will be saved in the local transaction list.

To process the command invocation request, the Communication Agent simply invokes the procedure specified by the given command ID number.

5.11 The Session Manager

The Session Manager helps users to establish connections with their shared users when they come to the shared environment. It also manages transactions and coordinates database updates for new users.

The Session Manager is transparent to VR programmers. All the local applications communicate with the Session Manager through their Communication Agents.

5.11.1 Shared User Management

The Session Manager processes the following two requests from local Communication Agents:

- Register the user
- Remove the user from the shared environment

The user registration function keeps a record of the new user's information including user name, host address, port number, the user's viewing places and viewing objects. The information is used to compare with other shared users' information to find out which users are the new user's coworkers. The coworkers' information is then sent to the new user; while the new user's information is sent to the coworkers. These coworkers will be instructed to update the database for the new user.

The user removal function removes the user from the user list. If any of the changes made at the user's site do not have coworkers any more, the corresponding transactions will be transferred to the Session Manager and saved at the manager's site.

5.11.2 Transaction Management

To update a new user's database, the Session Manager has to find out which shared user, including itself, has the most up–to–date information for the new user. This information includes the transactions for the new user's viewing places and viewing objects. The most up–to–date animation and alwaysActive types of data are also needed to correctly initialize the new user's environment.

There may be more than one user having the needed information. Certainly every user who is in the shared environment has the most up–to–date data of alwaysActive type. In order not to send repeated transactions to the new user, the Session Manager goes through the new user's shared user list and decides which one is responsible for sending the transactions about which objects and places. In case these users do not have all the information, the Session Manager will find the remaining information from the transactions saved at its site and send them to the new user.

When a user leaves the environment, if there are no coworkers for any of the user's viewing places or viewing objects, the Session Manager will accept the corresponding transactions and use them to update a new user's database later on. If the user is the last one in the environment, the Session Manager also has to save the transactions of all the alwaysActive shared data.

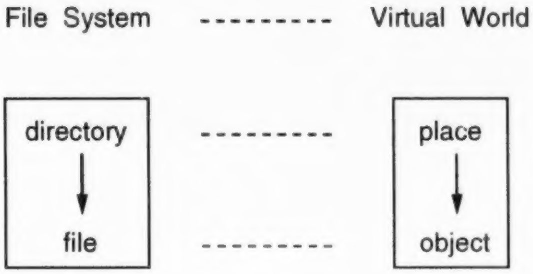

Figure 5.3: Analogy between Filesystem model and Place model

5.12 Extending Existing Computer Technologies

Looking back at the Filesystem model in the UNIX operating system environment, we find that there are many similarities between this model and the VR environment model. Since there are many well-developed technologies in conventional software development, we may borrow or extend them to get their counterparts for virtual reality technology. In this section, we extend the Filesystem model for the UNIX operating system and the software on top of it to produce a VR software system model.

5.12.1 The Filesystem Model as an Analogy to the Place Model

Figure 5.3 shows the correspondence between the Filesystem model and the Place model. In the Filesystem model, the file system refers to the UNIX operating system environment; in the Place model, virtual world refers to the whole VR environment. Directories and virtual places have the same characteristics and are used for the same purpose. Directories are special files while virtual places are special virtual objects. Directories look exactly like files from their parent directories. Virtual places are just boundary objects if we look at them from the outside. Nevertheless, they can contain

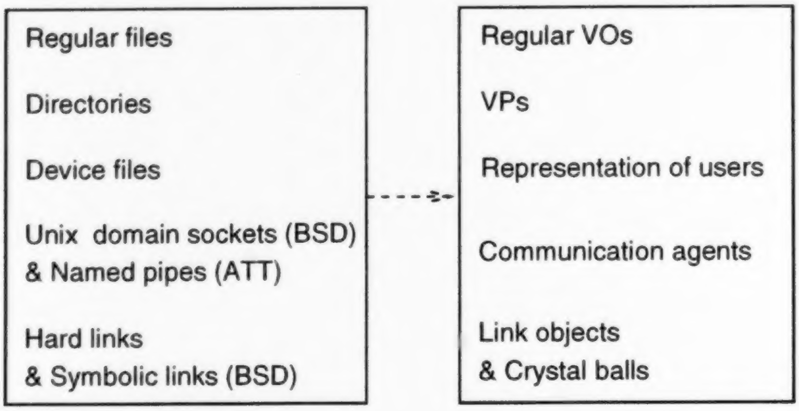

Figure 5.4: Correspondence between all types of files and VOs

the basic elements of their models — files and virtual objects, which in turn may be other directories and virtual places. In this way, they can be built into hierarchical structures and used effectively to organize environments. Files and virtual objects are the "meta" elements in the two environments to some extent. Everything in the operating system environment is a file whereas everything in the virtual world is a virtual object.

Examining the files and VOs more carefully, we can see that different types of files and VOs still have similar semantic meanings. There are seven types of files in UNIX operating systems. Their correspondence to the types of different VOs is shown in Figure 5.4. The most common type of file is a regular file whose content is just data. And the most common type of VOs is a regular VO, which is usually an application-defined VO. A directory is another type of file, while a VP is another type of VO as discussed before. Input/output data is made available through device files in the UNIX operating system. The same thing occurs in the virtual world. VR devices, such as the Polhemus digitizer, EyePhone and DataGlove, are all represented by a special type of compound virtual object — users. Unix domain sockets (BSD) and named pipes (ATT) are used for interprocess communications. Communication agents are responsible for communicating with the Session Manager and other users in the shared environment. Hard links and symbolic links are essentially the same from the user's perspective. They provide a copy of another file. The link objects and crystal balls serve the same purposes as their counterparts do in UNIX systems. Actually the idea of providing copies of objects and places originated from the idea of symbolic links for files. This borrowed technique can provide users with access to objects that are not in the users' current place. A link object is another display of the original VO. A crystal ball provides another display of the original place, and a navigation channel between the crystal ball's current place and its original place.

The Place model and the file system model are similar to each other in structure as well as in operations, such as create, move, remove (files, objects), and going from one place or directory to another place or directory. The similarities give us an indication of the kinds of operations a VR software system should support.

However, there are a few differences between the Virtual World and the file system. These differences include the structural difference and the difference in feedback resulting from the characteristics of an OS and VR.

In a virtual world, objects have hierarchical structure because sometimes operations on a whole object, as well as sub–objects, must be provided. But in a file system, a file is not a hierarchical structure itself. We need extra functions to deal with object structures.

Another difference is that the virtual world environment is an animation world, while the file system is a text–based environment. For example, we use "cd" to navigate around in the file system. When the "cd" command finishes, the prompt pops up and the user is in the new directory. No other feedback is provided, although the "pwd" command can be used to assure that the user is in the right directory. In a virtual world environment, the user issues a navigation command by making a gesture or selecting a command from a menu panel. What he or she expects is that a door will gradually open, and he or she is moved through the door from the current position to the other side of the door. If the user is suddenly transformed from the current place to another place, he or she could become confused. The user may not know

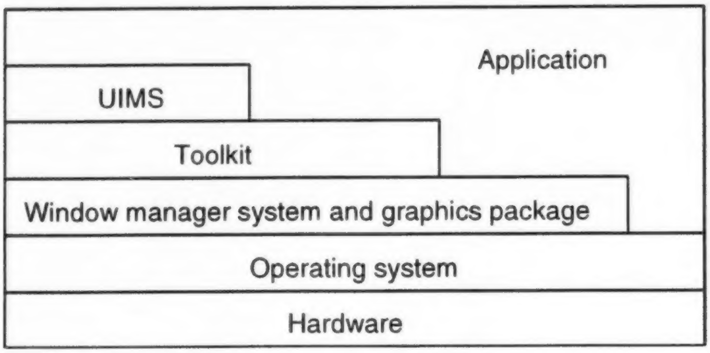

Figure 5.5: Traditional software system model

that the navigation command has finished. Therefore, the animated visual, auditory, or tactile feedback must be given in a virtual world environment. Every VO in the VW can have behaviors. Every behavior is actually an animation process. All active behaviors must be invoked frame after frame. In addition, many VOs' behaviors are initiated by the user. Usually, the user will be watching the process of the animation. Some processes can also be processes of direct manipulation. For instance, the user is dragging an object.

5.12.2 Extending the Traditional User Interface Model

We know that the file system is the environment model provided by the UNIX operating system for application programmers. And the UNIX system is the lowest level software that most programmers usually work with. To make the current software more friendly to users as well as to programmers, there are window manager systems, graphics packages, toolkits, and user interface management systems on top of the UNIX operating system. The traditional software system model is shown in Figure 5.5 (from [14]).

Because of the similarities between the structure of the virtual world and the file system, it is possible to build a VR software system to provide similar functionality for VR environments as the UNIX operating system provides with the file system.

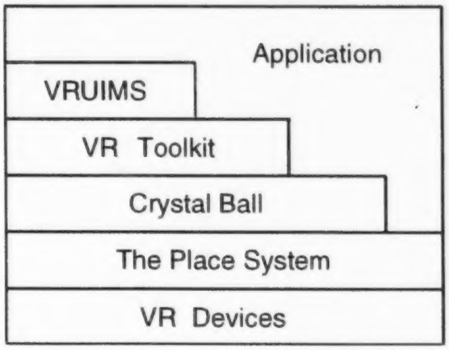

Figure 5.6: VR software system model

Naturally, we would like to extend the upper level software in traditional software system models too. Figure 5.6 shows the corresponding extension from the traditional software system model to the VR software system model.

Corresponding to the Hardware level in Figure 5.5, VR devices from the bottom level associated with servers and probably some software packages providing better services to the higher levels.

The operating system is a resource management system according to the file system environment model. Every hardware device looks like a file to the programmer. The operating system manages the static files and the running processes – the started executable object files. In comparison, the Place system does a lot of the operating system work, like managing places and objects in a VR environment and managing the behaviors of objects.

The window manager system in Figure 5.5 actually provides the user with a multipresence capability. The user can open several windows at the same time. He or she might be in different directories and doing different work in each window. The user can switch amongst windows by just moving the cursor from the current window to another window. The user can work very efficiently in this multipresence way. We can even find examples of multipresence in a big factory. In a central control room of the factory, there are many control monitors on which the situations in different workplaces can be seen. The crystal ball in the VR software system model gives the user a multipresence ability within the virtual world. The user can use many crystal balls through which the situation in other places of interest are displayed. The user can watch what is happening in other places and jump into a crystal ball, if he or she wishes to. Then the user is in that place. He or she can look back into the original place or return to that place later.

Above the Crystal ball level is the VR Toolkit. A set of tools will greatly improve programming efficiency in the development of applications. In the VR Toolkit, different interaction techniques can be provided. We can have different kinds of VPs such as rooms with doors, multiple floor buildings and parks. Transport facilities like elevators and helicopters can be provided. Clocks and watches can be used to report time to the user within virtual environments.

The top level of the software system is the VR User Interface Management System (VRUIMS). Just like a UIMS, a VRUIMS is a mechanism to start the application, call the functions provided by the programmer, keep the program running. This part will speed up the development of applications.

5.13 Summary

This chapter presents the Multi-User Place system, an implementation of the Multi-User VR system architecture described in the previous chapter. This Place system shows that the Virtual Object Model, Virtual Place Model, and the Multi-User VR system architecture do provide a feasible solution to the problem of response time improvement and network traffic reduction. The next chapter will evaluate how well the problem can be solved by the Place system.

The extension of traditional user interface model to VR is also discussed.

Chapter 6

Evaluation

The research goal of this thesis is to explore new VR techniques and provide high-level support to help VR programmers build high-performance, multi-user virtual environments. In this chapter, system performance, including response time and network traffic, is analyzed for the Place application systems. An example application is used as the basis for an experimental evaluation. The response time and network traffic are measured to see whether they achieve the expected performance.

6.1 Performance Analysis

The Multi-User Place System provides facilities to improve VR applications' response time and reduce the network traffic.

Typically, an application environment is composed of application objects. Let vw be the set of objects in an application environment without using the Place System, and VW be the objects in the same environment constructed using the Place System.

If the application environment is constructed using the Multi-User Place System, each application object can be a "meta" object and/or a hierarchy of sub-objects. By "meta" object I mean an object that has its own behaviors, rather than just a composition of its sub-objects. For example, the clock, which is the super-object of the frame, the dial markers, the hour hand, the minute hand and the second hand, is a "meta" object because it has an animation behavior of retrieving the current time from its host computer besides invoking the animation behaviors of the hour hand, minute hand and second hand.

So we have

$$object = \{OBJECT\} \cup \{subobj \mid subobj \text{ is a sub-object of } object\}$$

where $OBJECT$ is a "meta" object.

The application environment can be a place hierarchy. Each place has its boundary objects (composed of sub-objects of the place), contains application objects, and may have sub-places.

$$place = boundary \cup object \cup subPlace$$

where

$$\begin{aligned} boundary &= \{obj \mid obj \text{ is a boundary object of } place\} \\ object &= \{obj \mid obj \text{ is an application object in } place\} \\ subPlace &= \{subplc \mid subplc \text{ is a sub-place in } place\} \end{aligned}$$

For instance, the house includes the set of its boundary objects, the clock, the chair, the sofa, as well as the bedroom and the kitchen, which include their boundary objects and the objects inside them.

6.1.1 Response Time Improvement

The response time t_{rsp} of vw is the total response time of all application objects.

$$t_{rsp}(vw) = \sum_{obj \in vw} t_{rsp}(obj)$$

The response time of an application object is mostly due to the evaluation times of the object's event-handling behavior t_{evt}, ordinary animation behavior t_{ord} and always active animation behavior t_{alw}, as well as rendering time t_{rnd}.

$$t_{rsp}(object) = t_{evt}(object) + t_{ord}(object) + t_{alw}(object) + t_{rnd}(object)$$

So we have

$$t_{rsp}(vw) = \sum_{obj \in vw} (t_{evt}(obj) + t_{ord}(obj) + t_{alw}(obj) + t_{rnd}(obj))$$

For the corresponding Place application system, the behavior evaluation time and rendering time are denoted by T_{evt}, T_{ord}, T_{alw}, and T_{rnd}.

Event–Handling Behavior Evaluation

$$\begin{aligned} T_{evt}(object) &= \sum_{obj \in object} T_{evt}(obj) \\ T_{evt}(place) &= \sum_{obj \in boundary} T_{evt}(obj) + \sum_{obj \in object} T_{evt}(obj) + \sum_{obj \in subPlcB} T_{evt}(obj) \end{aligned}$$

where

$$subPlcB = \{boundary \text{ of } subplc | subplc \in subPlace\}$$

The *subPlcB* of the house is set of boundary objects of the bedroom and the kitchen.

Suppose the current place the user is in is:

$$curPlace = cpB \cup cpObj \cup cpSubP$$

where *cpB* is the set of boundary objects of *curPlace*, *cpObj* is the set of application objects in *curPlace*, and *cpSubP* is the set of sub–places in *curPlace*. We have

$$T_{evt}(VW) = T_{evt}(curPlace)$$

Therefore,

$$\begin{aligned} \frac{T_{evt}(VW)}{t_{evt}(vw)} &= \frac{T_{evt}(curPlace)}{t_{evt}(vw)} \\ &= \frac{\sum_{obj \in cpB} T_{evt}(obj) + \sum_{obj \in cpObj} T_{evt}(obj) + \sum_{obj \in cpSubPlcB} T_{evt}(obj)}{\sum_{obj \in vw} t_{evt}(obj)} \end{aligned}$$

where

$$cpSubPlcB = \{boundary \text{ of } subplc | subplc \in cpSubP\}$$

Assume

$$T_{evt}(object) = t_{evt}(object) \text{ where } object \text{ is an application object}$$

and

$$\sum_{obj \in cpB} T_{evt}(obj) + \sum_{obj \in cpSubPlcB)} T_{evt}(obj) \leq c_{curPlace} \sum_{obj \in cpObj} T_{evt}(obj)$$

where $c_{curPlace}$ is the smallest integer which satisfies the above inequality.

In case of the house, the event–handling behavior evaluation time of the boundary objects of the house, bedroom and kitchen should be less than or equal to the event–handling behavior evaluation time of the clock, chair and sofa. So $c_{house} = 1$. For places which have more application objects with more complicated event–handling behaviors, $c_{house} = 1$ is definitely enough.

Now we have

$$\frac{T_{evt}(VW)}{t_{evt}(vw)} \leq \frac{(c_{curPlace} + 1) \times \sum_{obj \in cpObj} T_{evt}(obj)}{\sum_{obj \in vw} t_{evt}(obj)}$$

If the application objects are equally distributed in every place and their $T_{evt}(object)$ are all the same,

$$\frac{T_{evt}(VW)}{t_{evt}(vw)} \leq \frac{c_{curPlace} + 1}{N_{place}}$$

where N_{place} is the number of places in VW. Therefore, using the Place system to organize large, complicated application environments is quite efficient in saving objects' event–handling behavior evaluation time.

In very simple application environments where all application objects are put in a single place, the VW, there are no sub–places. So

$$\sum_{obj \in cpSubPlcB} T_{evt}(obj) = 0$$

And there is no need to have boundary objects or say T_{evt} of the boundary objects are 0,

$$\sum_{obj \in cpB} T_{evt}(obj) = 0$$

Then we have

$$\begin{aligned}\frac{T_{evt}(VW)}{t_{evt}(vw)} &= \frac{\sum_{obj \in cpObj} T_{evt}(obj)}{\sum_{obj \in vw} t_{evt}(obj)} \\ &= 1\end{aligned}$$

As a result, using the Place system for building single place application environment is as efficient as without using it in event-handling behavior evaluation.

Ordinary Animation Behavior Evaluation

$$\begin{aligned}T_{ord}(object) &= \sum_{obj \in object} T_{ord}(obj) \\ T_{ord}(place) &= \sum_{obj \in boundary} T_{ord}(obj) + \sum_{obj \in object} T_{ord}(obj) + \sum_{plc \in subPlace} T'_{ord}(plc)\end{aligned}$$

where

$$T'_{ord}(place) = \begin{cases} T_{ord}(place) & \text{if } place \text{ is open} \\ \sum_{obj \in boundary} T_{ord}(obj) & \text{otherwise} \end{cases}$$

For example, $T'_{ord}(house)$ is equal to $T_{ord}(house)$ because the house is an open place and the user can see the inside of the house if he or she is in the virtual world. Whereas $T'_{ord}(kitchen)$ is equal to the animation behavior evaluation time of the boundary objects of the kitchen because the user can only see these boundary objects from outside of the kitchen.

There are two special types of objects whose ordinary animation behavior evaluation are more than just the evaluation of the object itself. One is the door object as a boundary object of the place. Whenever the door is open, the evaluation time is not only T_{ord} of the door object, but also T_{ord} of the top of the open places of the place on the other side of the door (See definition of "top of the open places" in previous chapter, section 5.5).

Let

$$door = \{DOOR\} \cup \{subobj | subobj \text{ is a sub-object of } door\}$$

and $topNeighborP$ be the top of the open places of the neighbor place on the other side of the open door.

$$T_{ord}(door) = \begin{cases} \sum_{obj \in door} T_{ord}(obj) + T_{ord}(topNeighborP) & \text{if } door \text{ is open} \\ \sum_{obj \in door} T_{ord}(obj) & \text{otherwise} \end{cases}$$

When the user is in the kitchen and the door to the house is open,

$$T_{ord}(kitchenDoor) = \sum_{obj \in kitchenDoor} T_{ord}(obj) + T_{ord}(virtualWorld)$$

The default door policy is to open the door only when the user is within a certain distance of the door. Most of the time at most one door is open for the local user. In this way, $T_{ord}(topNeighborP)$ can be saved.

Another special type of object is a crystal ball. There may be one or more crystal balls in a place. Each crystal ball introduces the ordinary behavior evaluation of its original place.

Let

$$crystalBall = \{CRYSBALL\} \cup \{originPlace\}$$

where $CRYSBALL$ is the crystal ball drawn outside of the original place and $originPlace$ is its original place.

We have

$$T_{ord}(crystalBall) = T_{ord}(originPlace)$$

As we can see from $T_{ord}(door)$ and $T_{ord}(place)$, open places consume more ordinary animation behavior evaluation time. By default all places are closed. In this case, $topNeighborP$ will be $neighborP$ which is the neighbor place itself. And

$$\begin{aligned} T'_{ord}(place) &= \sum_{obj \in boundary} T_{ord}(obj) \\ T_{ord}(place) &= \sum_{obj \in boundary} T_{ord}(obj) + \sum_{obj \in object} T_{ord}(obj) + \sum_{obj \in subPlcB} T_{ord}(obj) \end{aligned}$$

where $subPlcB = \{boundary \text{ of } subPlc | subPlc \in subPlace\}$.

Let $cpTopPlace$ be the top of the open places of the user's $curPlace$,

$$cpTopPlace = cpTopB \cup cpTopObj \cup cpTopSubP$$

where *cpTopB* is the set of the boundary objects of *cpTopPlace*, *cpTopObj* is the set of application objects in *cpTopPlace*, and *cpTopSubP* is the set of sub-places in *cpTopPlace*. We have

$$\begin{aligned}
T_{ord}(VW) &= T_{ord}(cpTopPlace) \\
\frac{T_{ord}(VW)}{t_{ord}(vw)} &= \frac{T_{ord}(cpTopPlace)}{t_{ord}(vw)} \\
&= \frac{\sum_{obj \in cpTopB} T_{ord}(obj) + \sum_{obj \in cpTopObj} T_{ord}(obj) + \sum_{plc \in cpTopSubP} T'_{ord}(plc)}{\sum_{obj \in vw} t_{ord}(obj)}
\end{aligned}$$

When all places are closed,

$$\begin{aligned}
cpTopPlace &= curPlace \\
\frac{T_{ord}(VW)}{t_{ord}(vw)} &= \frac{T_{ord}(curPlace)}{t_{ord}(vw)} \\
&= \frac{\sum_{obj \in cpB} T_{ord}(obj) + \sum_{obj \in cpObj} T_{ord}(obj) + \sum_{obj \in cpSubPlcB} T_{ord}(obj)}{\sum_{obj \in vw} t_{ord}(obj)}
\end{aligned}$$

Usually boundary objects do not have ordinary animation behaviors except opening doors. Plus, there is at most one door open at a time according to the default door policy. So

$$\sum_{obj \in cpB} T_{ord}(obj) + \sum_{obj \in cpSubPlcB} T_{ord}(obj) - T_{ord}(door) = 0$$

Therefore,

$$\begin{aligned}
\frac{T_{ord}(VW)}{t_{ord}(vw)} &= \frac{\sum_{obj \in cpObj} T_{ord}(obj) + T_{ord}(door)}{\sum_{obj \in vw} t_{ord}(obj)} \\
&= \frac{\sum_{obj \in cpObj} T_{ord}(obj) + \sum_{obj \in door} T_{ord}(obj) + T_{ord}(neighborP)}{\sum_{obj \in vw} t_{ord}(obj)}
\end{aligned}$$

Suppose there are N_{crys} number of crystal balls as objects in the user's *curPlace*,

$$\frac{T_{ord}(VW)}{t_{ord}(vw)} = \frac{\sum_{obj \in cpObj-crys} T_{ord}(obj) + \sum_{plc \in originP} T_{ord}(plc) + \sum_{obj \in door} T_{ord}(obj) + T_{ord}(neighborP)}{\sum_{obj \in vw} t_{ord}(obj)}$$

If all application objects are equally distributed in places, and each object has the same T_{ord}, every place has the same ordinary animation behavior evaluation time $T_{ord}(P)$. Then we have

$$\begin{aligned}
\sum_{obj \in cpObj-crys} T_{ord}(obj) + \sum_{obj \in door} T_{ord}(obj) &= T_{ord}(P) \\
\sum_{plc \in originP} T_{ord}(plc) &= N_{crys} \times T_{ord}(P) \\
T_{ord}(neighborP) &= T_{ord}(P) \\
\sum_{obj \in vw} t_{ord}(obj) &= N_{place} \times T_{ord}(P)
\end{aligned}$$

So

$$\frac{T_{ord}(VW)}{t_{ord}(vw)} = \frac{2 + N_{crystal}}{N_{place}}$$

However even when all places are open, we have the mechanism to prevent any repeated behavior evaluation. In the worst case,

$$\frac{T_{ord}(VW)}{t_{ord}(vw)} = 1$$

Always Active Animation Behavior Evaluation

$$\begin{aligned}
T_{alw}(object) &= \sum_{obj \in object} T_{alw}(obj) \\
T_{alw}(place) &= \sum_{obj \in boundary} T_{alw}(obj) + \sum_{obj \in object} T_{alw}(obj) + \sum_{plc \in subPlace} T_{alw}(plc) \\
T_{alw}(VW) &= \sum_{obj \in VW} T_{alw}(obj) + \sum_{obj \in boundary \text{ of } places} T_{alw}(obj)
\end{aligned}$$

where *places* is the set of places in VW.

Boundary objects don't have always active animation behaviors unless specified by VR programmers. So

$$\sum_{obj \in boundary \text{ of } places} T_{alw}(obj) = 0$$

Therefore,

$$T_{alw}(place) = \sum_{obj \in VW} T_{alw}(obj)$$

$$\begin{aligned}
\frac{T_{alw}(VW)}{t_{alw}(vw)} &= \frac{\sum_{obj \in VW} T_{alw}(obj)}{\sum_{obj \in vw} t_{alw}(obj)} \\
&= 1
\end{aligned}$$

Rendering

Similar to the ordinary animation behavior evaluation,

$$\begin{aligned}
T_{rnd}(object) &= \sum_{obj \in object} T_{rnd}(obj) \\
T_{rnd}(place) &= \sum_{obj \in boundary} T_{rnd}(obj) + \sum_{obj \in object} T_{rnd}(obj) + \sum_{plc \in subPlace} T'_{rnd}(plc)
\end{aligned}$$

where

$$T'_{rnd}(place) = \begin{cases} T_{rnd}(place) & \text{if } place \text{ is open} \\ \sum_{obj \in boundary} T_{rnd}(obj) & \text{otherwise} \end{cases}$$

$$\begin{aligned}
T_{rnd}(door) &= \begin{cases} \sum_{obj \in door} T_{rnd}(obj) + T_{rnd}(topNeighborP) & \text{if } door \text{ is open} \\ \sum_{obj \in door} T_{rnd}(obj) & \text{otherwise} \end{cases} \\
T_{rnd}(crystalBall) &= T_{rnd}(CRYSBALL) + T_{rnd}(originPlace)
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\frac{T_{rnd}(VW)}{t_{rnd}(vw)} \\
&= \frac{T_{rnd}(cpTopPlace)}{t_{rnd}(vw)} \\
&= \frac{\sum_{obj \in cpTopB} T_{rnd}(obj) + \sum_{obj \in cpTopObj} T_{rnd}(obj) + \sum_{plc \in cpTopSubP} T'_{rnd}(plc)}{\sum_{obj \in vw} t_{rnd}(obj)}
\end{aligned}$$

When all places are closed,

$$\begin{aligned}
T_{rnd}(door) &= T_{rnd}(DOOR) + T_{ord}(neighborP) \\
T'_{rnd}(place) &= \sum_{obj \in boundary} T_{rnd}(obj) \\
T_{rnd}(place) &= \sum_{obj \in boundary} T_{rnd}(obj) + \sum_{obj \in object} T_{rnd}(obj) + \sum_{obj \in subPlcB} T_{rnd}(obj)
\end{aligned}$$

So

$$\begin{aligned}\frac{T_{rnd}(VW)}{t_{rnd}(vw)} &= \frac{T_{rnd}(curPlace)}{t_{rnd}(vw)} \\ &= \frac{\sum_{obj\in cpB} T_{rnd}(obj) + \sum_{obj\in cpObj} T_{rnd}(obj) + \sum_{obj\in cpSubPlcB} T_{rnd}(obj)}{\sum_{obj\in vw} t_{rnd}(obj)}\end{aligned}$$

Suppose

$$\sum_{obj\in cpB} T_{rnd}(obj) + \sum_{obj\in cpSubPlcB} T_{rnd}(obj) = \sum_{obj\in cpObj} T_{rnd}(obj)$$

We have

$$\begin{aligned}&\frac{T_{rnd}(VW)}{t_{rnd}(vw)} \\ &= \frac{2 \times \sum_{obj\in cpObj-crys} T_{ord}(obj) + \sum_{plc\in originP} T_{ord}(plc) + T_{rnd}(neighborP)}{\sum_{obj\in vw} t_{rnd}(obj)} \\ &= \frac{3 + N_{crys}}{N_{place}}\end{aligned}$$

Best Case

$$\begin{aligned}T_{rsp}(VW) &= T_{evt}(VW) + T_{ord}(VW) + T_{alw}(VW) + T_{rnd}(VW) \\ &= T_{evt}(curPlace) + T_{ord}(cpTopPlace) + T_{alw}(VW) + T_{rnd}(cpTopPlace)\end{aligned}$$

The best case is when the following three conditions are all satisfied at the same time.

- There are no crystal balls.
- There are no open places.
- All doors are closed

We now have

$$\begin{aligned}T_{rsp}(VW) &= T_{evt}(curPlace) + T_{ord}(curPlace) + T_{alw}(VW) + T_{rnd}(curPlace) \\ &= T_{rsp}(curPlace) + T_{alw}(VW - curPlace)\end{aligned}$$

Because

$$T_{rsp}(curPlace) = T_{rsp}(cpB \cup cpSubPlcB) + T_{rsp}(cpObj)$$

Suppose

$$T_{rsp}(cpB \cup cpSubPlcB) = T_{rsp}(cpObj)$$

which means the response time of the boundary objects of the user's $curPlace$ and its sub–places are the same as the response time of the application objects in $curPlace$. This is a conservative assumption when the application environment has enough application objects in each place.

As to $T_{alw}(VW - curPlace)$, VR programmers can distribute the time–consuming computation of all always active animation behaviors to different computers and send the results to the users' hosts periodically. In this case,

$$T_{alw}(VW - curPlace) \ll T_{rsp}(curPlace)$$

As a result,

$$\begin{aligned} \frac{T_{rsp}(VW)}{t_{rsp}(vw)} &\approx \frac{2 \times T_{rsp}(cpObj)}{N_{place} \times T_{rsp}(cpObj)} \\ &\approx \frac{2}{N_{place}} \end{aligned}$$

Worst Case

The worst case is when all places in VW are open.

$$T_{rsp}(VW) = T_{evt}(curPlace) + T_{ord}(VW) + T_{alw}(VW) + T_{rnd}(VW)$$

In this case, only event–handling behavior evaluation time can be saved. Always active animation behavior evaluation time will remain about the same; while ordinary animation behavior evaluation time and rendering time will be increased by the amount of time of evaluating and displaying boundary objects respectively.

$$T_{evt}(VW) \;=\; T_{evt}(curPlace)$$

$$\begin{aligned} T_{ord}(VW) &= t_{ord}(vw) + \sum_{obj \in placeB} T_{ord}(obj) \\ T_{alw}(VW) &= t_{alw}(vw) \\ T_{rnd}(VW) &= t_{rnd}(vw) + \sum_{obj \in placeB} T_{rnd}(obj) \end{aligned}$$

where $placeB = \{boundary \text{ of } place | place \text{ is in } VW\}$. If the time spent on evaluating and displaying the boundary objects is more than the time saved from event-handling behavior evaluation, i.e.

$$\sum_{obj \in placeB} (T_{ord}(obj) + T_{rnd}(obj)) > t_{evt}(vw) - T_{evt}(curPlace)$$

The overall response time will be greater than the system response time without using the Place system. That is,

$$\frac{T_{rsp}(VW)}{t_{rsp}(vw)} > 1$$

6.1.2 Network Traffic Reduction

We will consider the number of transactions transmitted among shared users. Let $nt(vw)$ be the total number of transactions transmitted over the network if the Multi-User Place System is not used to build vw, and let $NT(VW)$ be the total number of transactions if the Multi-User Place System is used in the construction of VW. All transactions are initiated by users' actions, such as entering the shared environment, making changes in the environment, switching places, or leaving the environment. Let $nt(user)$ and $NT(user)$ denote the number of transactions initiated by the $user$'s actions. Now we have

$$\begin{aligned} nt(vw) &= \sum_{user \in shareUser} nt(user) \\ NT(VW) &= \sum_{user \in shareUser} NT(user) \end{aligned}$$

where $shareUser = \{user \mid user \text{ is in the shared environment } vw \text{ or } VW\}$.

When the user enters vw or a place in VW, related transactions must be transferred to the user to initialize his or her database. After the environment starts running, whenever the user makes changes, the results should be broadcast to the coworkers. When the user exits from vw or any place in VW, some transactions are needed for bookkeeping. So

$$\begin{aligned} nt(user) &= nt_{init}(user) + nt_{run}(user) + nt_{exit}(user) \\ NT(user) &= NT_{init}(user) + NT_{run}(user) + NT_{exit}(user) \end{aligned}$$

Environment Initialization

In vw, all objects have to be updated to initialize the environment correctly for the new user. But no initialization is needed when the user switches places. While in VW, when the user enters his or her first place and whenever the user switches places, the user's viewing objects, objects that are in the user's viewing places, and objects that have the AlwaysActive type of shared data have to be updated. We represent these objects by $updateSet_i$ when the user is in his or her ith place in VW, i.e.,

$$updateSet_i = viewSet_i \cup shareSet$$

where

$$\begin{aligned} viewSet_i &= \{obj \mid obj \text{ is a viewing object of the user who is in the } i\text{th place}\} \cup \\ &\quad \{obj \mid obj \text{ is in a viewing place of the user who is in the } i\text{th place}\} \\ shareSet &= \{obj \mid obj \text{ has the AlwaysActive type of shared data}\} \end{aligned}$$

We have

$$\begin{aligned} nt_{init}(user) &= \sum_{obj \in vw} nt_{init}(obj) \\ NT_{init}(user) &= \sum_{i=1}^{N_{curPlace}} \sum_{obj \in updateSet_i} NT_{init}(obj) \end{aligned}$$

where $nt_{init}(obj) = NT_{init}(obj)$ for all application objects, $nt_{init}(obj)$ and $NT_{init}(obj)$ are the number of shared data items in obj that have been updated, and $N_{curPlace}$ is the number of places the user has been in.

Suppose application objects are uniformly distributed in the places in VW and the average number of viewing places is $N_{viewPlace}$, the number of objects in $viewSet_i$ is $N_{viewPlace}$ times of the number of objects in one place. Assume the number of objects in $shareSet$ is about the same number of objects in one place, we have

$$\frac{NT_{init}(user)}{nt_{init}(user)} = \frac{N_{curPlace} \times (1 + N_{viewPlace})}{N_{place}}$$

where N_{place} is the number of places in VW.

Environment Updating

Once the environment starts to run, the local changes have to be sent to shared users to ensure a consistent view of the environment. In vw, the changes have to be sent to all other shared users, because it is impossible to tell the users who can or can not see the changes. However, in VW, if an object is modified, the result has to be sent to only those users who can see the object. If we use $coworker(obj)$ to represent the coworkers of obj, that is,

$$coworker(obj) = \{user \mid user \in shareUser \text{ and } obj \in viewSet_i \text{ of the } user\}$$

where $1 \leq i \leq N_{curPlace}$, we have

$$\begin{aligned} nt_{run}(user) &= \sum_{obj \in vw} N_{shareUser} \times nt_{run}(obj) \\ NT_{run}(user) &= \sum_{obj \in viewSet_i} N_{coworker}(obj) \times NT_{run}(obj) \end{aligned}$$

where $N_{shareUser}$ is the number of shared users in the environment and $N_{coworker}(obj)$ is the number of $coworker$s of the obj.

If the Animation or AlwaysActive types of shared data of an $object$ is modified, the result is saved but not sent to anyone else. Thus the transaction does not count in $NT_{run}(object)$. Because ordinary animation behaviors and always active animation behaviors most likely modify the above two types of shared data, most of $NT_{run}(obj)$ result from event–handling behavior evaluation. vw could use the same policy so that

$nt_{run}(obj)$ only counts transactions resulting from obj's event-handling behaviors. But we can still assume,

$$\sum_{obj \in vw} nt_{run}(obj) \geq \sum_{obj \in viewSet_i} NT_{run}(obj)$$

Suppose every object in VW has an average number of coworkers $N_{coworker}$, then

$$\begin{aligned} \frac{NT_{run}(user)}{nt_{run}(user)} &= \frac{N_{coworker} \times \sum_{obj \in viewSet_i} NT_{run}(obj)}{N_{shareUser} \times \sum_{obj \in vw} nt_{run}(obj)} \\ &\leq \frac{N_{coworker}}{N_{shareUser}} \end{aligned}$$

Environment Termination

When the $user$ quits from vw, if he or she is the last one in the shared environment, all transactions must be transferred to somewhere and saved for new users. If there are other users in vw, no transactions need to be saved. Thus

$$nt_{exit}(user) = \begin{cases} \sum_{obj \in vw} nt_{exit}(obj) & \text{if } shareUser - \{user\} = \{\} \\ 0 & \text{otherwise} \end{cases}$$

On average, the probability of $user$ being the last user in the environment is $1/N_{shareUser}$. Therefore,

$$nt_{exit}(user) = \frac{\sum_{obj \in vw} nt_{exit}(obj)}{N_{shareUser}}$$

When the $user$ leaves a place in VW, the transactions of the objects that do not have any other coworkers have to be transferred to the Session Manager. If the $user$ is leaving VW and he or she is the last user in the shared environment, the AlwaysActive type of shared data have to be saved by the Session Manager too. Now we have

$$NT_{exit}(user) = \sum_{i=1}^{N_{curPlace}} \sum_{obj \in viewSet_i} NT'_{exit}(obj) + \sum_{obj \in shareSet} NT'_{exit}(obj)$$

For $obj \in viewSet_i$,

$$NT'_{exit}(obj) = \begin{cases} NT_{exit}(obj) & \text{if } coworker(obj) - \{user\} = \{\} \\ 0 & \text{otherwise} \end{cases}$$

The probability of $user$ being the last coworker of obj is $1/N_{coworker}(obj)$. So

$$NT'_{exit}(obj) = \frac{NT_{exit}(obj)}{N_{coworker}(obj)}$$

If the numbers of $coworkers$ for all objs are the same, say $N_{coworker}$,

$$NT'_{exit}(obj) = \frac{NT_{exit}(obj)}{N_{coworker}}$$

As a result,

$$\sum_{obj \in viewSet_i} NT'_{exit}(obj) = \frac{\sum_{obj \in viewSet_i} NT_{exit}(obj)}{N_{coworker}}$$

Similarly, for $obj \in shareSet$,

$$\begin{aligned} NT'_{exit}(obj) &= \begin{cases} NT_{exit}(obj) & \text{if } shareUser - \{user\} = \{\} \\ 0 & \text{otherwise} \end{cases} \\ &= \frac{NT_{exit}(obj)}{N_{shareUser}} \\ \sum_{obj \in shareSet} NT'_{exit}(obj) &= \frac{\sum_{obj \in shareSet} NT_{exit}(obj)}{N_{shareUser}} \end{aligned}$$

So,

$$NT_{exit}(user) = \frac{\sum_{i=1}^{N_{curPlace}} \sum_{obj \in viewSet_i} NT_{exit}(obj)}{N_{coworker}} + \frac{\sum_{obj \in shareSet} NT_{exit}(obj)}{N_{shareUser}}$$

We have

$$\begin{aligned} \frac{NT_{exit}(user)}{nt_{exit}(user)} &= \frac{\frac{\sum_{i=1}^{N_{curPlace}} \sum_{obj \in viewSet_i} NT_{exit}(obj)}{N_{coworker}} + \frac{\sum_{obj \in shareSet} NT_{exit}(obj)}{N_{shareUser}}}{\frac{\sum_{obj \in vw} nt_{exit}(obj)}{N_{shareUser}}} \\ &= \frac{N_{shareUser}}{N_{coworker}} \times \frac{\sum_{i=1}^{N_{curPlace}} \sum_{obj \in viewSet_i} NT_{exit}(obj)}{\sum_{obj \in vw} nt_{exit}(obj)} + \\ &\quad \frac{\sum_{obj \in shareSet} NT_{exit}(obj)}{\sum_{obj \in vw} nt_{exit}(obj)} \end{aligned}$$

where $nt_{exit}(obj) = NT_{exit}(obj)$ for all application objects. $nt_{exit}(obj)$ and $NT_{exit}(obj)$ are the number of shared data items in obj that have been updated.

Suppose the application objects are equally distributed in all places, the number of objects in $viewSet_i$ are $N_{viewPlace}$ times of objects in one place. Thus

$$\frac{\sum_{i=1}^{N_{curPlace}} \sum_{obj \in viewSet_i} NT_{exit}(obj)}{\sum_{obj \in vw} nt_{exit}(obj)} = \frac{N_{curPlace} \times N_{viewPlace}}{N_{place}}$$

where $N_{viewPlace}$ is the average number of viewing places, and N_{place} is the number of places in the application environment.

Assume the number of objects that have the AlwaysActive type of shared data is about the same number of objects in one place, then

$$\frac{\sum_{obj \in shareSet} NT_{exit}(obj)}{\sum_{obj \in vw} nt_{exit}(obj)} = \frac{1}{N_{place}}$$

Now

$$\frac{NT_{exit}(user)}{nt_{exit}(user)} = \frac{N_{shareUser}}{N_{coworker}} \times \frac{N_{curPlace} \times N_{viewPlace}}{N_{place}} + \frac{1}{N_{place}}$$

Best Case

The best case happens when no user has other viewing places besides the current place, and every user stays in the current place long enough before switching to the next place so that the cost of initiating and terminating places can be ignored compared to the number of transactions at run time. In an environment where no user has other viewing places, $N_{coworker}$ is reduced to the number of users sharing the same current place. That every user remaining in the current place long enough will reduce the overhead of initiating and terminating the user's places and improve the efficiency of the shared environment.

Suppose $NT(user)$ and $nt(user)$ are the same for every user,

$$\begin{aligned}
\frac{NT(VW)}{nt(vw)} &= \frac{\sum_{user \in shareUser} NT(user)}{\sum_{user \in shareUser} nt(user)} \\
&= \frac{N_{shareUser} \times NT(user)}{N_{shareUser} \times nt(user)} \\
&= \frac{NT(user)}{nt(user)} \\
&= \frac{NT_{init}(user) + NT_{run}(user) + NT_{exit}(user)}{nt_{init}(user) + nt_{run}(user) + nt_{exit}(user)}
\end{aligned}$$

Since

$$\begin{aligned}
NT_{init}(user) + NT_{exit}(user) &\ll NT_{run}(user) \\
nt_{init}(user) + nt_{exit}(user) &\ll nt_{run}(user)
\end{aligned}$$

So

$$\begin{aligned}\frac{NT(VW)}{nt(vw)} &= \frac{NT_{run}(user)}{nt_{run}(user)}\\ &= \frac{\sum_{obj\in viewSet_i} N_{coworker} \times NT_{run}(obj)}{\sum_{obj\in vw} N_{shareUser} \times nt_{run}(obj)}\\ &= \frac{N_{coworker} \times \sum_{obj\in viewSet} NT_{run}(obj)}{N_{shareUser} \times \sum_{obj\in vw} nt_{run}(obj)}\end{aligned}$$

Assume there are equal number of objects in each place in VW, and $NT_{run}(obj)$ and $nt_{run}(obj)$ are all the same for every object,

$$\frac{NT(VW)}{nt(vw)} = \frac{N_{coworker}}{N_{shareUser}} \times \frac{1}{N_{place}}$$

Worst Case

The worst case is when all users can see every place in VW and they are switching from places to places all the time while making very few changes. We have,

$$updateSet_i = \{obj \mid obj \in VW\}$$

So

$$NT_{init}(user) = N_{curPlace} \times nt_{init}(user)$$

For $NT_{run}(user)$,

$$\begin{aligned}N_{coworker} &= N_{shareUser}\\ viewSet_i &= \{obj \mid obj \in VW\}\end{aligned}$$

Then

$$NT_{run}(user) = nt_{run}(user)$$

As to $NT_{exit}(user)$, because

$$\begin{aligned}viewSet_i &= \{obj \mid obj \in VW\}\\ shareSet &\subset viewSet_i\\ N_{coworker} &= N_{shareUser}\end{aligned}$$

we have

$$NT_{exit}(user) = N_{curPlace} \times nt_{exit}(user)$$

Because

$$\begin{aligned}\frac{NT(VW)}{nt(vw)} &= \frac{\sum_{user \in shareUser} NT(user)}{\sum_{user \in shareUser} nt(user)} \\ &= \frac{\sum_{user \in shareUser}(NT_{init}(user) + NT_{run}(user) + NT_{exit}(user))}{\sum_{user \in shareUser}(nt_{init}(user) + nt_{run}(user) + nt_{exit}(user))}\end{aligned}$$

and we assume that all users switches places very often but make few modifications,

$$\begin{aligned}NT_{run}(user) &\ll NT_{init}(user) + NT_{exit}(user) \\ nt_{run}(user) &\ll nt_{init}(user) + nt_{exit}(user)\end{aligned}$$

So we have

$$\frac{NT(VW)}{nt(vw)} = N_{curPlace}$$

6.2 Application Example

An example two–user application has been built using the Multi–User Place System to demonstrate the system's functionality. It is a two–user application, with one user using the Eyephone and Dataglove and the other user using the ADL-1 and Polhemus Isotrak for the user's eye and hand position and orientation. The same two–user example application has also been built without using the Multi–User Place System. In this section, the application built using the Multi–User Place is refered to as the PlaceApplication, while the application that does not use the Place system is refered to as the NoPlaceApplication. The response time and network traffic of the two applications is compared with each other to see if the response time is improved and if the network traffic is reduced for the PlaceApplication.

6.2.1 Multi–User Place System Application

The example application environment, used from chapter 3 to chapter 5, is constructed according to the Place model. Figure 6.1 shows the hierarchical structure of the application environment.

In the example environment, there are two sub–places (the kitchen and the bedroom) in the house which is an open place in the virtual world. The clockLink object in the bedroom is a link object of the clock that is in the house, while the ovenCrystal is a crystal ball of the oven that is in the kitchen.

The source code for this environment is given in appendix A.1.

Some of the application objects, such as donuts, are simple objects. Others, like clock and chair, are composed of sub–objects. Figure 6.2 shows the hierarchical structure of the clock object.

The clock is an object that reports the current time. The ordinary animation behavior of the clock retrieves the time from its host computer, informs the hour hand, the minute hand and the second hand about the current time, then invokes their ordinary animation behaviors to adjust their angles so as to display the time. The code for the clock is in appendix A.2.

In order to allow the users to re–arrange the position of an object, the standard VObject class in the Place system is extended to an AObject class. The event–handling behaviors of the AObject class tests whether it is grabbed by the local user's hand, and translates according to the hand's movement if grabbed. The application objects, including the clock, chair, sofa, table, and bed, are all instances of the AObject class.

Whenever an object of class AObject is grabbed and moved to a new position, the other user should be informed of the transformation, if it is possible for him or her to see the object. In the constructor of the AObject class, the object matrix is registered with the local Communication Agent. When the object is moved, the data updating function is invoked to send the updated matrix to the other user.

Appendix A.3 shows the source code for class AObject.

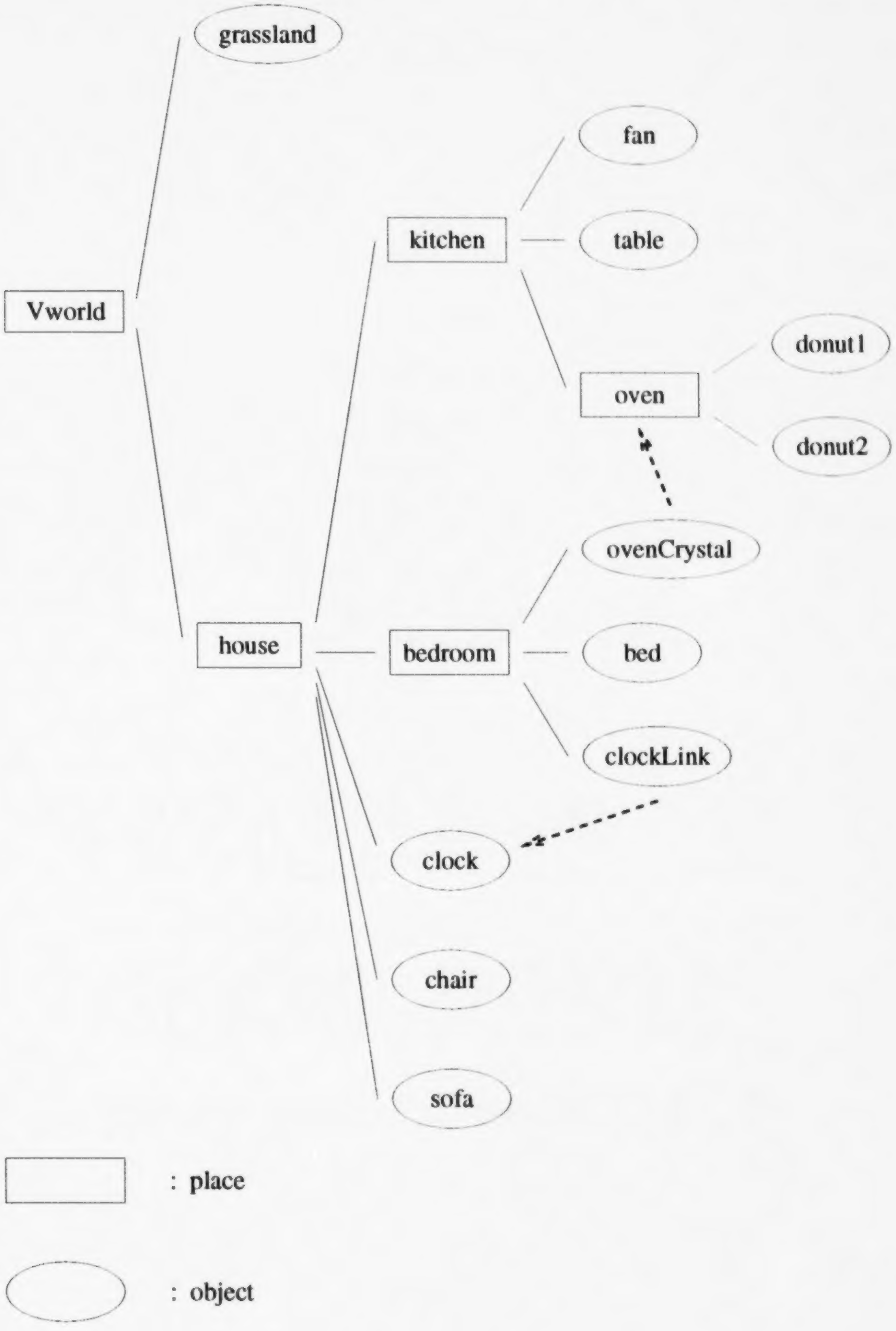

Figure 6.1: Hierarchical structure of the application environment

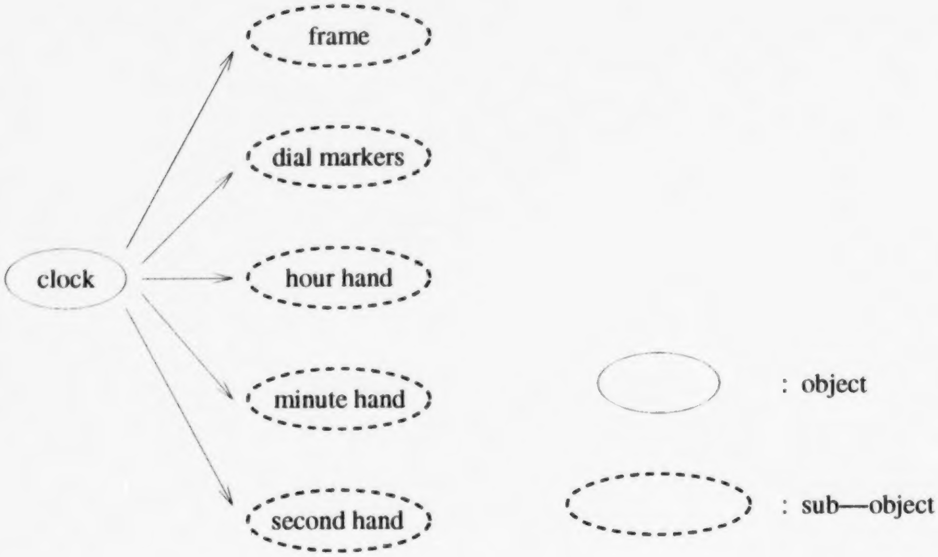

Figure 6.2: Hierarchical structure of the clock object

6.2.2 Response Time Improvement

The same two-user example application has been built without using the Multi-User Place system. This is similar to SIMNET [5], DIVE [7], and BrickNet [37] with same contents for both users, the event-handling behaviors and ordinary animation behaviors of all objects are evaluated. All the objects in the example environment are rendered, whether they are visible to the local user or not. If one user drags an object to a new position, the object's updated transformation matrix is sent to the other user even though the object may not be visible to that user. It is not appropriate to build the house environment using BrickNet with the two clients, userA and userB, in different worlds. First, if userA's world is the kitchen, and userB's world is the house plus the outside of the house, while nobody is in the bedroom, it is not clear where the server can find all the objects in the bedroom to lease them to userB when userB wants to go into the bedroom. Second, there is no intuitive navigational facility with animation feedback available for the users to go from one place to another. The users have to select a menu item to be taken to another world using a "dynamic portal" mechanism.

In the tests, userB enters the kitchen first, userA then enters the house and starts to move the chair around until the program quits after 1000 updates. The time spent on communication (com), evaluation of event-handling behaviors (evt), ordinary animation behaviors (ord), and always active animation behaviors (alw), and rendering (rnd), as well as the update rate are measured for userA and userB in both applications. The test results for the PlaceApplication are then compared with the results for the NoPlaceApplication to determine the response time improvement for userA and userB.

The communication time can be hard to determine if the time that the two users share the environment is different. The longer the two users share the environment, the longer the communication time is. The event-handling behavior evaluation time varies according to the users' activities in the environment. If a user is always moving an object around, this user's event-handling behavior evaluation time is longer. The ordinary animation behavior evaluation time is about the same for every test in this house environment. As we will see, the always active animation behavior evaluation time increases as the update rate of the application increases, while the update rate is mostly determined by the rendering time, which increases as the user looks at more and complex objects, instead of few and simple polygonal objects.

In order to show that the response time is improved for both users in even the worst cases, the two users always enter the environment at about the same time, and look at as many objects as possible. Because we don't have the dataGlove for userB, userB can not do the "grab" gesture to drag any objects around. But, userA always moves the objects around for as long as possible.

Table 6.1 and table 6.2 show 5 test results for userA and userB in the NoPlaceApplication; while table 6.3 and table 6.4 show the results in the PlaceApplication.

Figure 6.3 compares userA's average performance data in the NoPlaceApplication with its average performance data in the PlaceApplication, whereas figure 6.4 compares the average results in the two applications for userB.

test No.	com	evt	ord	alw	rnd	update rate
	(seconds)					(updates/second)
test 1	2.3	20.5	3.2	1.6	154.8	5.5
test 2	2.1	22.0	3.0	1.4	151.8	5.5
test 3	1.9	23.2	3.1	1.5	157.7	5.3
test 4	2.4	24.3	3.5	1.5	153.6	5.4
test 5	1.9	22.3	3.0	1.3	159.2	5.3
average	2.1	22.4	3.2	1.4	155.4	5.4

Table 6.1: Test results for userA in the NoPlaceApplication

test No.	com	evt	ord	alw	rnd	update rate
	(seconds)					(updates/second)
test 1	1.7	10.4	2.8	2.3	186.8	4.9
test 2	1.9	10.7	2.7	1.8	185.6	4.9
test 3	1.7	12.1	3.4	1.7	188.9	4.8
test 4	2.0	11.7	3.2	2.0	186.2	4.9
test 5	1.9	11.0	2.9	2.2	186.2	4.9
average	1.8	11.2	3.0	2.0	186.7	4.9

Table 6.2: Test results for userB in the NoPlaceApplication

test No.	com	evt	ord	alw	rnd	update rate
	(seconds)					(updates/second)
test 1	2.6	11.6	1.9	3.4	49.9	14.2
test 2	2.2	12.6	1.9	3.5	51.7	13.7
test 3	2.4	12.6	2.0	3.5	53.6	13.4
test 4	2.3	11.8	1.7	3.6	50.3	14.1
test 5	2.8	12.6	1.8	3.9	49.9	13.9
average	2.5	12.2	1.9	3.6	51.1	13.9

Table 6.3: Test results for userA in the PlaceApplication

test No.	com	evt	ord	alw	rnd	update rate
	(seconds)					(updates/second)
test 1	2.9	7.6	1.3	2.6	91.5	9.4
test 2	2.3	7.8	0.9	2.9	96.3	9.0
test 3	2.0	8.6	1.1	2.9	96.7	8.9
test 4	2.4	8.1	0.9	2.4	92.8	9.3
test 5	2.4	8.0	1.4	3.0	93.4	9.2
average	2.4	8.0	1.1	2.7	94.1	9.2

Table 6.4: Test results for userB in the PlaceApplication

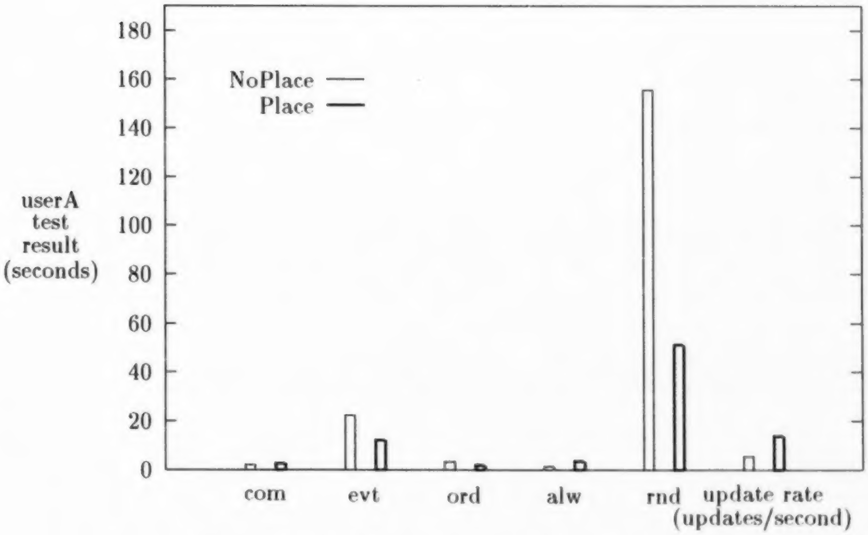

Figure 6.3: Compare userA's results in both applications

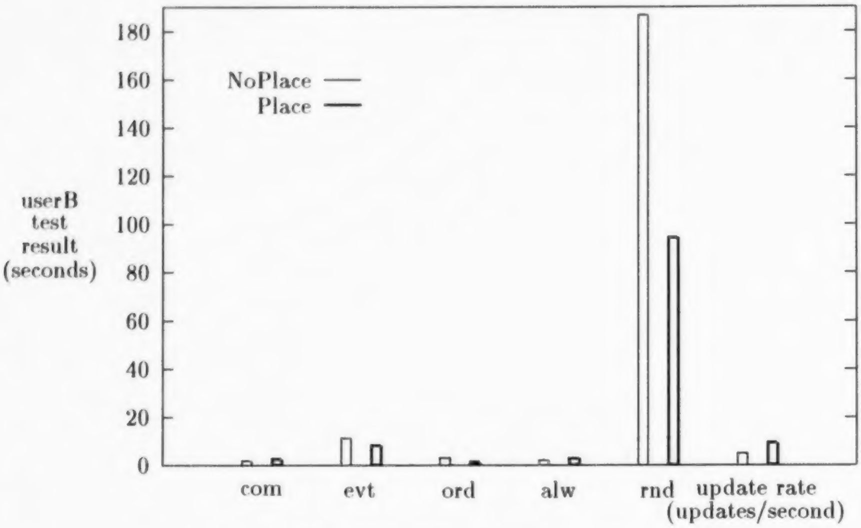

Figure 6.4: Compare userB's results in both applications

We can see that communication takes about the same amount of time for userA and userB in NoPlaceApplication. The communication time for the two users in PlaceApplication is more than the communication time in NoPlaceApplication because the transaction management is more complicated, and takes more time in PlaceApplication than in NoPlaceApplication when userA drags the chair around and the fan is running in the kitchen.

In both applications, userA uses much more event–handling behavior evaluation time than userB, because userA is moving the chair around and userB is not changing anything in the kitchen. UserA and userB do save some event–handling behavior evaluation time in PlaceApplication over the NoPlaceApplication, since only the objects in their current place are evaluated.

Comparing the ordinary animation behavior evaluation time for userA and userB in PlaceApplication with the time in NoPlaceApplication, PlaceApplication saves time since it only evaluates the clock's time–reporting behavior for userA and the fan blades' turning behavior for userB.

Only the two donuts in the example environment have always active baking behaviors. The baking behavior of each donut keeps changing the color of the donut

application	com	evt	ord	alw	rnd	update rate
without Place	2.1	22.4	3.2	1.4	155.4	5.4
with Place	2.5	12.2	1.9	3.6	51.1	13.9
improve (%)	-19.0	45.5	40.6	-157.1	67.1	61.2

Table 6.5: Compare the test results of userA in NoPlaceApplication and in PlaceApplication

application	com	evt	ord	alw	rnd	update rate
without Place	1.8	11.2	3.0	2.0	186.7	4.9
with Place	2.4	8.0	1.1	2.7	94.1	9.2
improve (%)	-33.3	28.6	63.3	-35.0	49.6	46.7

Table 6.6: Compare the test results of userB in NoPlaceApplication and in PlaceApplication

until the specified baking time (in real time) has passed. The higher the update rate is, the more frames in which the donuts' baking behaviors must be evaluated. As we can see, the update rates for both userA and userB are improved in PlaceApplication. Therefore, in this particular application, the always active behavior evaluation time for userA and userB in PlaceApplication is more than it is in NoPlaceApplication.

Rendering time is the most dominant factor of response time. As shown in figure 6.3 and figure 6.4, both userA and userB's rendering time in PlaceApplication are greatly improved because the PlaceApplication does not draw any objects in the kitchen or in the bedroom for userA, and only draws the objects in the kitchen for userB. The rendering for userB takes more time than the rendering for userA, since the two donuts are Non–Uniform Rational B–Spline (NURBS) surfaces which take more time to render than polygons.

The update rates for both users are improved in PlaceApplication mostly due to the rendering time improvement.

Table 6.5 compares the average performance data for userA in Table 6.1 and Table 6.3, while table 6.6 compares the average performance data for userB in Ta-

number of messages	establishing connection		updating databases		terminating connection		total	
	sent	recv'd	sent	recv'd	sent	recv'd	sent	recv'd
test 1	1	2	419	1	1	0	421	3
test 2	1	2	402	1	1	0	404	3
test 3	1	2	407	1	1	0	409	3
test 4	1	2	423	1	1	0	425	3
test 5	1	2	373	1	1	0	375	3

Table 6.7: Transactions occured at userA's site in NoPlaceApplication

ble 6.4 and in Table 6.2. We can see that the Multi–User Place system does help this example application save a lot of rendering time as well as some evaluation time of event–handling behaviors and ordinary animation behaviors. Consequently, the update rates are greatly improved for both userA and userB, although the communication time is a little more in the PlaceApplication than in the NoPlaceApplication and the always active behaviors of the donuts take more time because of higher update rates.

6.2.3 Network Traffic Reduction

The number of messages sent and received by the Session Manager (SM), userA and userB are also recorded in both applications to see if PlaceApplication can save any network traffic. For userA and userB, the number of messages are counted in the following three periods of time: when the users enter the environment, when they are interacting with the objects in the environment, and when they are ready to leave the environment.

Table 6.7 and table 6.8 give the number of messages sent and received by userA and userB in the NoPlaceApplication. The number of messages sent and received by SM is always 3 and 5 in the 5 tests.

Figure 6.5 shows the communication process in the NoPlaceApplication. The numbers along the arrows denotes the order of messages sent or received. Because

number of messages	establishing connection		updating databases		terminating connection		total	
	sent	recv'd	sent	recv'd	sent	recv'd	sent	recv'd
test 1	1	1	2	419	2	1	5	421
test 2	1	1	2	402	2	1	5	404
test 3	1	1	2	407	2	1	5	409
test 4	1	1	2	423	2	1	5	425
test 5	1	1	2	373	2	1	5	375

Table 6.8: Transactions occured at userB's site in NoPlaceApplication

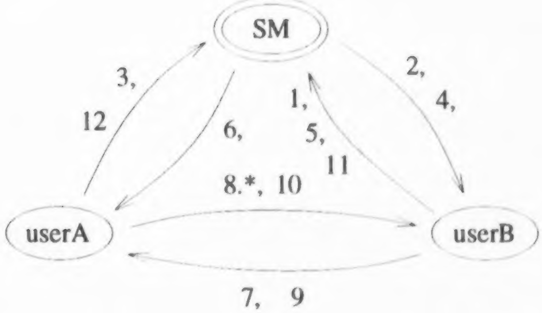

Figure 6.5: Communications among SM, userA and userB in NoPlaceApplication

userB always enters the environment first and the update rate for userA and userB are almost the same, userB always finishes the 1000 updates first and quits first from the environment. The following list explains each message:

1. UserB registers at SM site to enter the kitchen;
2. SM acknowledges userB's registration;
3. UserA registers at SM site to enter the house;
4. SM instructs userB to send the always active behavior data to userA;
5. UserB acknowledges the above message;
6. SM acknowledges userA's registration with userB's information;
7. UserB sends donuts' always active behavior data to userA;
8. UserA sends the chair's updated transformation matrix to userB;
9. UserB informs userA that he or she is leaving the environment;
10. UserA acknowledges the leaving message;
11. UserB informs SM that he or she is leaving the environment;
12. UserA informs SM of leaving the environment as well as sends the chair's transformation matrix and donuts' data to SM;

Table 6.9 and table 6.10 show the number of messages sent and received by userA and userB in the PlaceApplication.

Figure 6.6 illustrates the above case.

1. UserB registers at SM site to enter the kitchen;
2. SM acknowledges userB's registration;
3. UserA registers at SM site to enter the house;

number of messages	establishing connection		updating databases		terminating connection		total	
	sent	recv'd	sent	recv'd	sent	recv'd	sent	recv'd
test 1	1	2	1	1	2	1	4	4
test 2	1	2	1	1	2	1	4	4
test 3	1	2	1	1	2	1	4	4
test 4	1	2	1	1	2	1	4	4
test 5	1	2	1	1	2	1	4	4

Table 6.9: Transactions occured at userA's site in PlaceApplication

number of messages	establishing connection		updating databases		terminating connection		total	
	sent	recv'd	sent	recv'd	sent	recv'd	sent	recv'd
test 1	1	1	2	1	2	2	5	4
test 2	1	1	2	1	2	2	5	4
test 3	1	1	2	1	2	2	5	4
test 4	1	1	2	1	2	2	5	4
test 5	1	1	2	1	2	2	5	4

Table 6.10: Transactions occured at userB's site in PlaceApplication

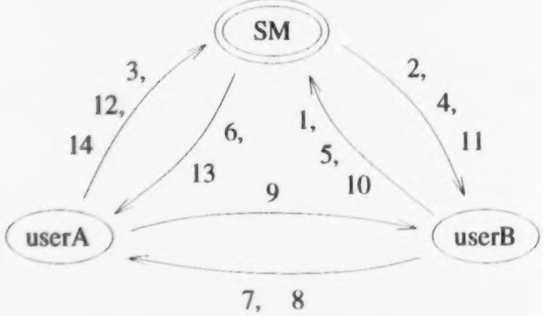

Figure 6.6: Communications among SM, userA and userB in PlaceApplication

4. SM instructs userB to send the always active behavior data to userA;
5. UserB acknowledges the above message;
6. SM acknowledges userA's registration with userB's information;
7. UserB sends donut's always active behavior data to userA;
8. UserB informs userA that he or she is leaving the environment;
9. UserA acknowledges the leaving message;
10. UserB informs SM that he or she is leaving the environment;
11. SM acknowledges the above message;
12. UserA informs SM of leaving the environment as well as sends the chair's transformation matrix to SM;
13. SM asks for the donuts' always active behavior data because userA is the last user in the shared environment;
14. UserA sends the donuts' data to SM;

As we can see, even though the PlaceApplication requires a few more messages for maintaining the transactions when userA quits from the shared environment, it significantly reduces network traffic by not sending the chair's transformation matrix when userA moves the chair around in the house.

6.3 Summary

The theoretical evaluation of a Place application system reveals that crystal balls, open doors and especially open places are very expensive in terms of response time and network traffic, because they make more places visible to the users. Although the event–handling behavior evaluation time and always–active behavior evaluation time are not affected by the number of places visible to a user, the ordinary animation behavior evaluation time and rendering time, as well as network traffic are affected dramatically by the number of visible places. Therefore, from the performance point of view, a Place application system programmer should use crystal balls, open doors and open places as little as possible.

In the example house environment, a crystal ball was used in the bedroom to allow a user to view the donuts being baked in the oven. Since neither userA nor userB were in the bedroom, the experimental results are not affected by the existence of this crystal ball. Furthermore, the doors were always closed during the experiments. Therefore, open doors are not a factor considered in the results. However, the place userA is in is an open place because it has a window through which userA can see the outside of the house. Since there is only one object outside of the house, the grassland having no behaviors and little rendering cost, the fact that the house is an open place should not have a major effect on the response time for userA, and have no effect on the network traffic between userA and userB.

Taking into account that not all the application objects are evenly distributed in the house environment and that the behavior evaluation time and rendering time for the objects are not the same, the experimental result for the example application do not exactly match the theoretical evaluation results that are derived under those assumptions. However, the result of the example application did demonstrate that the Place system indeed improves the event-handling evaluation time, the ordinary animation evaluation time, as well as the rendering time, and therefore improves the overall response time. The total number of transactions is also greatly reduced even though some transactions are needed for initializing and terminating the shared environment.

Rendering time is the most dominant factor in this example application. Further experiments where network traffic is the dominant factor need to be conducted. For example, a virtual environment may contain a complex object, such as a soft ball which is composed of hundreds of thousands of small polyhedrons. Whenever the user squishes the ball, the transformation matrices for all the polyhedrons must be sent to the shared users, resulting in heavy network traffic. When shared users are geographically distributed to different sites, the network delay will become a major problem which should be further investigated as well.

Chapter 7

Conclusion

7.1 Contributions

The main contribution of this thesis is the introduction of the Virtual Object Model, the Virtual Place Model, and the Multi-User VR System Architecture. The Virtual Object Model is proposed to assist programmers to analyze, design and implement VR application systems from a high level to a more detailed level. The Virtual Place Model is proposed to organize application environments into hierarchical structure so we can take advantage of this environment structure to improve response time and reduce network traffic for VR applications. The Multi-User VR System Architecture allows shared users to join or quit from a shared environment without affecting each other.

The Multi-User Place System was built to support the proposed models. It is a framework that can integrate application objects into VR environments. The application objects are specified as in the Virtual Object Model. The VR environments are hierarchically structured as described in the Virtual Place Model. The navigation facilities, such as doors and flying, enable the users to move around in the virtual environment. Link objects and the crystal ball mechanism provide accesses to non-current places, which relieves the disadvantages of putting application objects in different places. Behavior culling and visible object rendering both contribute to the improvement of the system response time. The response time of a Place application, either a single user application, or a multi–user application, is not proportional to the complexity of the whole virtual environment, but proportional only to the complexity of the current place the user is in.

A multi–user VR application system can benefit even more by using the Multi–User Place System. The built–in Session Manager and Communication Agents support database synchronization among shared users and automatic database update for late–comers. The naming mechanism for dynamically created shared data is improved so that multiple users can create objects at the same time without competing for their internal names. The updated shared data is sent only to the users who can see the changes so that the network traffic is reduced. The number of messages sent and received by a user is proportional only to the number of users who share the same places, as opposed to the number of all users who share the entire environment with the user.

None of the previous VR systems, including IBM rubber rocks, SIMNET, WAVES and BrickNet, used hierarchical structure in their application environments.

Using the Place system, the performance of a simple application environment like in IBM rubber rocks system should be as good as the original system. For systems like SIMNET, if the environment can be properly divided, the response time for each user as well as network traffic among shared users should be improved depending on the environment structure. WAVES and BrickNet application systems allow different users to be in different worlds. If each user only needs very small environment for their own, Place applications can work as well as WAVES and BrickNet. However, when any user needs a larger environment, and this environment can be further divided into small sub–environments, the performance of the Place application will exceed the performance of WAVES and BrickNet. The BrickNet system can also be used to build applications with different users in the same environment. In this case, the Place application will definitely perform better than the BrickNet application in terms of response time and at least as good as BrickNet application in terms of network traffic.

7.2 Future Work

In order to improve the functionality of the Multi-User Place System, the following problems need to be further studied: objects and places' security problem, concurrency control, multi-user system bottlenecks, and the traveller problem.

7.2.1 Security Problem

In a Multi-User Place application system, a user may have some objects and places which the user does not want other users to see or modify. In the Unix file system, the permission mode for a file or directory decides the read, write, and execute permissions of the shared users. Similarly, the permissions to view and modify an object or a place can be introduced to the Multi-User Place System. An object or a place should not be visible to a user who does not have view permission. If a user does not have modify permission on an object, he or she should not be allowed to manipulate the object. When a user registers to enter a place, the permission mode of the place should be checked to see if the user is allowed to be in that place.

7.2.2 Concurrency Control

When several users are cooperating on a project, they may all have permission to modify a particular object. A VR system must use a concurrency control mechanism to ensure a consistent copy of the object at all users sites.

One strategy that people working on Computer Aided Cooperative Work suggest is called "floor control". Only one user controls the floor at a time. Only the user who controls the floor can manipulate the shared object. After that user is done, the control is passed on to the next user. The floor control strategy could be implemented by granting the object modify permission only to the user who controls the floor. The permission must be changed whenever the control is passed on to another user.

This is a non–optimistic serialization approach which will affect the responsiveness of VR systems. On the other hand, optimistic serialization approaches will produce very confusing environments for the user to work in. Semi–optimistic approaches might be the way to go for VR application systems. They should provide better responsiveness than non–optimistic approches and are easier to implement and less confusing to users than the optimistic approaches.

7.2.3 System Bottleneck

The potential system bottleneck is the Session Manager, which is responsible for processing the user's request to enter the shared environment, to quit from the environment, and to switch places. When there are too many shared users, the number of requests may go beyond the processing power of the Session Manager. Consequently, all users have to wait for their requests to be processed. The Session Manager then becomes the system bottleneck.

Multiple session managers could be used to share the responsibilities of the current Session manager. Each session manager can be assigned to a specific set of places. If a user is always working within one set of places, the session manager which is responsible for this set of places can process all requests from that user. Otherwise, several session managers have to work together to serve the needs of this user.

7.2.4 Traveller Problem

By traveller, I mean a user who frequently moves from one place to another. If a Multi–User Place application system has a traveller, the Session Manager will receive "switch place" requests very often. The probability that the Session Manager becomes the system bottleneck is much higher than when no traveller is present in a shared environment. Moreover, all related shared users have to adjust their shared user list and update the database for the traveller whenever navigation takes place. Consequently, the performance for shared users can be affected because their Communication Agents spend more time to process the traveller's requests. The network traffic is not necessarily reduced either.

One possible solution is to provide an option for a user to claim to be a traveller when the user joins in the environment. The Multi–User Place System can treat the traveller as a coworker of every other shared user. In this way, neither the Session Manager nor any of the shared users need to be bothered when the traveller switches places. However, the network traffic produced because of the traveller will be the same as when the Multi–User Place System is not used.

Appendix A

Example Application Programs

A.1 Program of Example Environment Construction

```
#include "/usr/granada/grad/yunqi/include/VP/Vinit.h"
#include "/usr/granada/grad/yunqi/include/VP/Vdebug.h"
#include "/usr/granada/grad/yunqi/include/VP/Vrectangle.h"
#include "/usr/granada/grad/yunqi/include/VP/Vwall.h"
#include "/usr/granada/grad/yunqi/include/VP/VplainDoor.h"
#include "/usr/granada/grad/yunqi/include/VP/VwallWithDoor.h"
#include "/usr/granada/grad/yunqi/include/VP/VoldStyleWindow.h"
#include "/usr/granada/grad/yunqi/include/VP/VwallWithWindow.h"
#include "/usr/granada/grad/yunqi/include/VP/Vroom.h"
#include "/usr/granada/grad/yunqi/include/VP/Vlink.h"
#include "/usr/granada/grad/yunqi/include/VP/Vcrystal.h"
#include "include/Aclock.h"
#include "include/Atable.h"
#include "include/Achair.h"
#include "include/Asofa.h"
#include "include/Abed.h"
#include "include/Aoven.h"
#include "include/Atorus.h"
#include "include/Afan.h"

VRoom *house;
VRoom *bedroom;
VRoom *kitchen;

VWall* houseWalls[6] = {NULL, NULL, NULL, NULL, NULL, NULL};
```

```
void vwConstruct ();

int NEWSOFA;
void VCreateSofa ();

int main(int argc, char** argv) {

    VInit(argc, argv);

    vwConstruct ();

    NEWSOFA = VcomAgent->VNewCommand (house, VCreateSofa);

    short handDevice = VlocalHand->VGetHandDevice ();

    if (handDevice == 1) {
        VlocalUser->VEnterWorld(*house);
    } else {
           VlocalUser->VEnterWorld(*kitchen);
    }

    VExit ();
};

void vwConstruct () {

    Vworld->VSetSize (20.0, 10.0, 3.01);
    Vbackground->VSetRGBColor (120, 200, 255);

    VRectangle *vground = new VRectangle ("ground", 20.0, 10.0);
    vground->VSetRGBColor (100, 255, 100);
    Vworld->VAddSubObj(*vground);

    VPlainDoor *houseDoor = new VPlainDoor ("houseDoor");
    houseDoor->VTranslate (4.0, 0.0, 0.0);

    houseWalls[0] = new VWallWithDoor ("housefrontWall",
                                          10.0, 3.0, houseDoor);

    VOldStyleWindow *window = new VOldStyleWindow ("window");
    window->VTranslate (4.0, 0.0, 1.0);
    houseWalls[3] = new VWallWithWindow ("housebackWall", 10.0, 3.0, window);
```

```
house = new VRoom ("house", 10.0, 5.0, 3.0, houseWalls, 6);
house->VTranslate (5.5, 5.0, 0.01);
Vworld->VAddSubPlace (*house);

VClock *vclock = new VClock ("clock");
vclock->VTranslate (3.2, 4.99, 2.0);
vclock->VRotate(900, 'x');
house->VAddObj (vclock);

AChair *chair = new AChair ("chair");
chair->VTranslate (3.2, 4.4, 0.01);
house->VAddObj (chair);

bedroom = new VRoom ("bedroom", 4.0, 2.5, 3.0);
bedroom->VTranslate (6.0, 2.5, 0.01);
house->VAddSubPlace (*bedroom);

VLink *clockLink = new VLink ("clockLink", *vclock);
clockLink->VTranslate (1.0, 2.49, 2.0);
clockLink->VRotate(900, 'x');
clockLink->VScale (1.5, 1.5, 1.5);
bedroom->VAddObj (clockLink);

ABed *bed = new ABed ("bed");
bed->VTranslate (2.0, 2.49, 0.0);
bed->VRotate (2700, 'z');
bedroom->VAddObj (bed);

bedroom->VSetPortPos (1.0, 1.2, 0.0);

kitchen = new VRoom ("kitchen", 5.0, 2.5, 3.0);
kitchen->VTranslate (2.5, 0.0, 0.01);
kitchen->VRotate(900, 'z');
house->VAddSubPlace (*kitchen);

ATable *table = new ATable ("table");
table->VTranslate (3.0, 1.5, 0.01);
kitchen->VAddObj (table);

AOven *oven = new AOven ("oven", 0.6, 0.4, 0.4);
oven->VTranslate (3.2, 2.3, 0.62);
oven->VRotate (2700, 'z');
oven->VOpenPlace ();
```

```
    kitchen->VAddSubPlace (*oven);

    ATorus *donut1 = new ATorus ("donut1");
    donut1->VSetCookingTime (6.0);
    donut1->VTranslate (0.2, 0.15, 0.15);
    oven->VAddObj (donut1);

    ATorus *donut2 = new ATorus ("donut2");
    donut2->VSetCookingTime (9.0);
    donut2->VTranslate (0.4, 0.15, 0.15);
    oven->VAddObj (donut2);

    AFan *fan = new AFan ("fan");
    fan->VTranslate (4.0, 0.85, 0.01);
    fan->VRotate (2700, 'z');
    kitchen->VAddObj (fan);

    kitchen->VSetPortPos (2.5, 1.5, 0.0);

    VCrystal *crysOven = new VCrystal ("crysKitchen", oven);
    crysOven->VTranslate (1.0, 2.0, 1.0);
    bedroom->VAddObj (crysOven);

    house->VSetPortPos (5.5, 1.5, 0.0);
};

void VCreateSofa () {
    ASofa *newSofa = new ASofa ("newSofa");

    newSofa->VTranslate (5.3, 4.2, 0.01);
    newSofa->VRotate (2700, 'z');
    house->VAddObj (newSofa);
};
```

A.2 Program of Class VClock

A.2.1 Aclock.h

```
#include "/usr/granada/grad/yunqi/src/appl/include/Aobject.h"
```

```
#ifndef VCLOCK_H
#define VCLOCK_H

/***************************************
*   Class VClock                       *
***************************************/

class VClock : public AObject {
public:
    VClock(char*);
    ~VClock() {};
    virtual void VGenClkObj();
    virtual void VAnimation();
protected:
    struct timeval *tv;
    struct tm *vTime;
    int hourIndex;
    int minuteIndex;
    int secondIndex;
};

#endif
```

A.2.2 Aclock.C

```
#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <malloc.h>
#include <gl.h>
#include "/usr/granada/grad/yunqi/include/VP/Vinit.h"
#include "/usr/granada/grad/yunqi/include/VP/Vdebug.h"
#include "/usr/granada/grad/yunqi/include/VP/Vrectangle.h"
#include "include/Aobject.h"
#include "include/AdialMarker.h"
#include "include/AclockHand.h"
#include "include/Aclock.h"

VClock::VClock(char *clkName) : AObject(clkName) {

    vTime = (struct tm *)malloc(sizeof(struct tm));
    tv = (struct timeval *)malloc(sizeof(struct timeval));

    gettimeofday(tv,NULL);
    vTime = localtime(&(tv->tv_sec));

    VGenClkObj();
};

void VClock::VGenClkObj() {
    VRectangle *frame;
    VDialMarker *dialMarker;
    VClockHand *hourHand;
    VClockHand *minuteHand;
    VClockHand *secondHand;

    frame = new VRectangle("frame", 0.6, 0.6);
    frame->VTranslate(-0.3, -0.3, 0.0);
    frame->VSetRGBColor(200,200,200);
    VAddSubObj(*frame);

    VSetRGBColor(6,6,6);

    dialMarker = new VDialMarker ("dialMarker", 0.024, 0.03);
    dialMarker->VTranslate (0.0, 0.0, 0.01);
    VAddSubObj(*dialMarker);

    hourHand = new VClockHand ("hourHand", 0.032, 0.18);
    hourHand->VTranslate (0.0, 0.0, 0.01);
    hourIndex = subObjNum;
    VAddSubObj(*hourHand);

    minuteHand = new VClockHand ("minuteHand", 0.025, 0.28);
    minuteHand->VTranslate (0.0, 0.0, 0.01);
    minuteIndex = subObjNum;
    VAddSubObj(*minuteHand);

    secondHand = new VClockHand ("secondHand", 0.015, 0.30);
    secondHand->VTranslate (0.0, 0.0, 0.01);
    secondIndex = subObjNum;
    VAddSubObj(*secondHand);

};
```

```
void VClock::VAnimation() {
    int seconds, minutes, countermin, hours;
    int secang, minang, hourang;

    gettimeofday(tv,NULL);
    vTime = localtime(&(tv->tv_sec));

    seconds = vTime->tm_sec;
    secang = (3600 - seconds * 60) % 3600;  // sec/60 = ang/360
    ((VClockHand*)subObj[secondIndex])->VUpdateTime(secang);

    minutes = vTime->tm_min;
    countermin = minutes * 60;
    minang = (3600 - countermin) % 3600;
    ((VClockHand*)subObj[minuteIndex])->VUpdateTime(minang);

    hours = vTime->tm_hour % 12 ;
    hourang = (3600 - hours * 300 - countermin / 12) % 3600;
    ((VClockHand*)subObj[hourIndex])->VUpdateTime(hourang);

    for (int i = 2; i < subObjNum; i++) {
        subObj[i]->VAnimation ();
    }
};
```

A.3 Program of Class AObject

A.3.1 Aobject.h

```
#include "/usr/granada/grad/yunqi/include/VP/Vobject.h"

#ifndef AOBJECT_H
#define AOBJECT_H

/****************************************
*   Class AOBJECT                       *
****************************************/

class AObject : public VObject {
public:
    AObject(char*);
    ~AObject() {};
    virtual int AIsGrabbedByHand ();
    virtual void VEventHandler();
protected:
    int grabbed;    // 1: grabbed by hand; 0: released;
    int matrixId;   // dataId for object matrix;
};

#endif
```

A.3.2 Aobject.C

```
#include <gl.h>
#include "/usr/granada/grad/yunqi/include/VP/Vinit.h"
#include "/usr/granada/grad/yunqi/include/VP/Vdebug.h"
#include "include/Aobject.h"

AObject::AObject(char *AobjName) : VObject(AobjName) {

    grabbed = 0;
    matrixId = VcomAgent->VRegisterData (this, (char*) mat, sizeof (mat));
};

int AObject:: AIsGrabbedByHand () {
    float handPos[4];

    int ifGrabbing = VlocalHand->VIsGrabbing ();

    if ((grabbed == 0) && (ifGrabbing == 1)) {
         VlocalHand->VGetHandPos (handPos[0], handPos[1], handPos[2]);
         handPos[3] = 1.0;
         grabbed = VWithinBoundary (*VlocalHand, handPos, 0.05, 0.05, 0.05);
    } else {
        int ifFlat = VlocalHand->VIsFlat ();
        if ((grabbed == 1) && (ifFlat == 1)) {
            grabbed = 0;
        }
    }

    return (grabbed);
};
```

```
void AObject::VEventHandler () {
    int ifGrabbed;
    float prevHand[3];
    float curHand[3];
    float offHand[3];

    if ((superObj[0] == NULL) || (superObj[0] == VlocalHand)) {
        // if this is the top object;
        ifGrabbed = AIsGrabbedByHand ();
        if ((grabbed == 0) && (ifGrabbed == 1)) {
            curPlace->VRmObj (this);
            VlocalHand->VAddSubObj (*this);
        } else {
            if ((grabbed == 1) && (ifGrabbed == 0)) {
                VlocalHand->VRmSubObj (*this);
                curPlace->VAddObj (this);
            }
        }
    }

    if (grabbed == 1) {     // dragged by hand;
        VlocalHand->VGetPrevHandPos (prevHand[0], prevHand[1], prevHand[2]);
        VlocalHand->VGetHandPos (curHand[0], curHand[1], curHand[2]);
        offHand[0] = curHand[0] - prevHand[0];
        offHand[1] = curHand[1] - prevHand[1];
        offHand[2] = curHand[2] - prevHand[2];

        VTranslate (offHand[0], offHand[1], offHand[2]);

        VcomAgent->VUpdateData (matrixId);
    }

    VObject::VEventHandler ();
};
```

Bibliography

[1] Kenji Akiyama, Nobuji Tetsutani, Morito Ishibashi, Susumu Ichinose, and Hiroshi Yasuda. Consideration on three-dimensional visual communication systems. *IEEE Journal on Selected Areas in Communications*, 9(4):555–560, May 1991.

[2] Anthony J. Aretz. Spatial cognition and navigation. In *Proceedings of the Human Factors Society 33rd Annual Meeting*, volume 1, pages 8–12, October 1989.

[3] M.T. Bolas and S.S. Fisher. Head-coupled remote stereoscopic camera system for telepresence applications. In *Stereoscopic Displays and Applications, Proceedings of SPIE*, volume 1256, pages 113–123, February 1990.

[4] Steve Bryson and Scott S. Fisher. Defining, modeling, and measuring system lag in virtual environments. In *Stereoscopic Displays and Applications, Proceedings of SPIE*, volume 1256, pages 98–109, 1990.

[5] James Calvin, Alan Dickens, Bob Gaines, Paul Metzger, Dale Miller, and Dan Owen. The SIMNET virtual world architecture. In *IEEE Virtual Reality Annual International Symposium'93*, pages 450–455, 1993.

[6] Stuart K. Cark, George G. Robertson, and Jack D. Mackinlay. The information visualizer, an information workspace. In *CHI'91 Conference Proceedings*, pages 181–188, 1991.

[7] Christer Carlsson and Olof Hagsand. DIVE — a multi–user virtual reality system. In *IEEE Virtual Reality Annual International Synposium VRAIS'93*, pages 394–400, 1993.

[8] Christopher Codella, Reza Jalili, Lawrence Koved, J. Bryan Lewis, Daniel T. Ling, James S. Lipscomb, David A. Rabenhorst, and Chu P. Wang. Interactive simulation in a multi–person virtual world. In *CHI'92 Conference Proceedings*, pages 329–334, 1992.

[9] Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, and Raymond Tomlinson. Mmconf: An infrastructure for building shared multimedia applications. In *Proceedings of CSCW'90*, pages 329–342, 1990.

[10] T.J. Doll, J.M. Gerth, W.R. Engelman, and D.J. Folds. Development of simulated directional audio for cockpit applications. Technical Report AAMRL–TR–86–014, USAF, 1986.

[11] Steven Feiner and Clifford Beshers. Visualizing n–dimensional virtual worlds with n–vision. *Computer Graphics, Special issue on 1990 Symposium on Interactive 3D Graphics*, 24(2):37–38, March 1990.

[12] Steven Feiner and Clifford Beshers. Worlds within worlds metaphors for exploring n–dimensional virtual worlds. In *UIST'90, Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 76–83, October 1990.

[13] S.S. Fisher, M. McGreevy, J. Humphries, and W. Robinett. Virtual environment display system. In *Proceedings of the 1986 Workshop on Interactive 3–D Graphics*, pages 77–87, New York: Association for Computing Machinery, 1987.

[14] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics Principles and Practice*, page 436. Addison–Wesley Publishing Company, 2nd edition, 1990.

[15] James J. Gibson. *The Senses Considered as Perceptual Systems.* Houghton mifflin company, Boston, 1966.

[16] Saul Greenberg and Ralph Bohnet. Groupsketch: A multi–user sketchpad for geographically–distributed small groups. In *Graphics Interface'91*, pages 207–215, 1991.

[17] Saul Greenberg and David Marwood. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of CSCW'94*, pages 207–217, 1994.

[18] Kelly Harwood. Cognitive perspective on map displays for helicopter flight. In *Proceedings of the Human Factors Society 33rd Annual Meeting*, volume 1, pages 13–17, October 1989.

[19] Hiroo Iwata. Artificial reality with force–feedback: Development of desktop virtual space with compact master manipulator. *Computer Graphics*, 24(4):165–170, August 1990.

[20] M.G. Kaye, Judith Ineson, D.N. Jarrett, and G. Wickham. Evaluation of virtual cockpit concepts during simulated missions. In *Helmet–Mounted Display II, Proceedings of SPIE*, volume 1290, pages 236–245, April 1990.

[21] Rick Kazman. Making WAVES: On the design of architectures for low–end distributed virtual environments. In *IEEE Virtual Reality Annual International Synposium VRAIS'93*, pages 443–449, 1993.

[22] Myron W. Krueger. *Artificial Reality.* Addison–Wesley Publishing Company, 1983.

[23] Myron W. Krueger. *Artificial Reality II.* Addison–Wesley Publishing Company, 1991.

[24] J. Bryan Lewis, Lawrence Koved, and Daniel T. Ling. Dialogue structures for virtual worlds. In *CHI'91 Conference Proceedings*, pages 131–136, 1991.

[25] Jiandong Liang, Chris Shaw, and Mark Green. On temporal–spatial realism in the virtual reality environment. In *UIST'91, Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 19–25, November 1991.

[26] Jack D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: Detail and context smoothly integrated. In *CHI'91 Conference Proceedings*, pages 173–179, 1991.

[27] I.E. McDowall, M. Bolas, S. Pieper, S.S. Fisher, and J. Humphries. Implementation and integration of a counterbalanced crt–based stereoscopic display for interactive viewpoint control in virtual environment applications. In *Stereoscopic Displays and Applications, Proceedings of SPIE*, volume 1256, pages 136–146, February 1990.

[28] Margaret Minsky, Ming Ouh-young, Oliver Steele, Frederick P. Brooks, Jr., and Max Behensky. Feeling and seeing: Issues in force display. *Computer Graphics, Special Issue on 1990 Symposium on Interactive 3D Grphics*, 24(2):235–244, March 1990.

[29] Charles E. Mosher Jr., George W. Sherouse, Peter H. Mills, Keven L. Novins, Stephen M. Pizer, Julian G. Rosenman, and Edward L. Chaney. The virtual simulator. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*. ACM, 1987.

[30] Ming Ouh-young, Michael Pique, John Hughes, Neela Srinivasan, and Frederick P. Brooks, Jr. Using a manipulator for force display in molecular docking. In *IEEE Robotics and Automation Conference Proceedings*, volume 3, pages 1824–1829, April 1988.

[31] Nicholas J.M. Patrick, Thomas B. Sheridan, and Michael Massimino. Design and testing of a non–reactive, fingertip, tactile display for interaction with remote environments. In *Cooperative Intelligent Robotics in Space, Proceedings of SPIE*, volume 1387, pages 215–222, 1990.

[32] D.R. Perrott and J. Tucker. Minimum audible movement angle as a function of signal frequency and the velocity of the source. *Journal of the Acoustical Society of America*, 83:1522–1527, 1988.

[33] Robert K. Rebo and Phil Amburn. A helmet–mounted virtual environment display system. In *Helmet–Mounted Displays, Proceedings of SPIE*, volume 1116, pages 80–84, 1989.

[34] George G. Robertson, Stuart K. Card, and Jack D. Mackinlay. The cognitive coprocessor architecture for interactive user interface. In *UIST'89, Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 10–18, 1989.

[35] Warren Robinett and Jannick P. Rolland. A computational model for the stereoscopic optics of a head–mounted display. Technical Report TR91–009, The University of North Carolina at Chapel Hill, Department of Computer Science, February 1991.

[36] Chris Shaw, Jiandong Liang, Mark Green, and Yunqi Sun. The decoupled simulation model for virtual reality systems. In *CHI'92 Conference Proceedings*, pages 321–328, 1992.

[37] Gurminder Singh, Luis Serra, Willie Png, and Hern Ng. BrickNet: A software toolkit for network–based virtual worlds. *PRESENCE: Teleoperators and Virtual Environments (to appear).*

[38] Ivan E. Sutherland. A head–mounted three dimensional display. In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 757–764. Washington: The Thompson Book Company, 1968.

[39] VPL Research Inc. *DATAGLOVE GESTURE EDITOR SOFTWARE For the Apple Macintosh.* Redwood City, California, March 1989.

[40] VPL Research Inc. *DATAGLOVE MODEL 2 Operation Manual.* Redwood City, California, August 1989.

[41] VPL Research Inc. *EYEPHONE Operation Manual.* Redwood City, California, June 1989.

[42] Chu P. Wang, Lawrence Koved, and Semyon Dukach. Design for interactive performance in a virtual laboratory. *Computer Graphics, Special Issue on 1990 Symposium on Interactive 3D Grphics*, 24(2):39–40, March 1990.

[43] E. M. Wenzel, F. L. Wightman, D. J. Kistler, and S.H. Foster. Acoustic origins of individual differences in sound localization behavior. *Journal of the Acoustical Society of America*, 84, S79, 1988.

[44] Elizabeth M. Wenzel and Scott H. Foster. Realtime digital synthesis of virtual acoustic environments. *Computer Graphics, Special Issue on 1990 Symposium on Interactive 3D Grphics*, 24(2):139–140, 1990.

[45] Elizabeth M. Wenzel, Frederic L. Wightman, and Doris J. Kistler. Localization with non-individualized virtual acoustic display cues. In *CHI'91 Conference Proceedings*, pages 351–359, 1991.

[46] D. Zeltzer, X. Pieper, and D. Sturman. An integrated graphical simulation platform. In *Proceedings of Graphics Interface'89*, pages 266–274, 1989.

[47] M. Zientara. New view on the world – pioneer designs 3d camera lenses with micro. *InfoWorld*, May 1984.